Universidad Carlos III de Madrid

Escuela Politécnica Superior

Ingeniería Informática

Proyecto de Fin de Carrera

Madrid, Julio 2013

Evaluación y desarrollo de módulo hardware en SoC programable

Autor: Javier Castrejón Torrejón

Tutores: Óscar Pérez Alonso

Resumen

En la actualidad, el mercado de los dispositivos de bajo y medio consumo avanza a una velocidad de vértigo, debido al gran aumento de la escala de integración y las nuevas tecnologías desarrolladas en este tipo de dispositivos.

Una de las tecnologías más sonadas en la actualidad en el mercado de dispositivos de medio y bajo consumo se trata de la tecnología del **System On Chip**. Una tecnología que integra todos los componentes de un sistema de computación completo como bloques de un solo chip. Esta tecnología permite una reducción **costes de producción**, **consumo**, **tamaño** y aumentando en gran medida su **capacidad de procesamiento**.

Los System On Chip están compuestos por diferentes **bloques** que representan las diferentes partes de un computador. A principio de 2012, Xilinx la mayor fabricante de dispositivos programables, lanza el Zynq-7000 un System on Chip programable.

El Zynq-7000 se trata de un System on Chip que contiene una **lógica programable** similar a una FPGA, y que por lo tanto permite reprogramar parte de su hardware, para adaptarse a las necesidades de los clientes.

Con este proyecto se quiere realizar una **investigación** y **evaluación** del Zynq-7000 y de sus herramientas de desarrollo. Además se realizará una **implementación** y **prueba** de un **módulo hardware**, y se implementará en un sistema HW del chip propio. Finalmente se desarrollará una **plataforma de trabajo** para permitir el desarrollo **software** sobre la plataforma creada.

El principal objetivo con la realización de este proyecto de investigación es que sea un **punto de partida** para poder desarrollar un proyecto hardware o software adaptado a las necesidades de un cliente.

Abstract

Currently, the market of devices low to medium advances at a lightning speed, due to the large increase in the scale of integration and the new technologies developed in such devices.

Currently one of the most talked about technologies on the market for these devices is the technology of **System On Chip**. This technology integrates all components of a complete computer system as blocks of a single chip. This technology **lowers costs** of **production**, **consumption**, size and **greatly increases** performance.

System On Chip devices made up of different **blocks** corresponding to the components of a computer. In early 2012, the largest manufacturer of Xilinx programmable devices, the Zynq-7000 releases an programmable System on Chip.

Zynq-7000 is a System on Chip with a **programmable logic** like an FPGA and allows reprogramming part of its hardware, to suit the needs of customers.

With this project carry out **research** and **evaluation** of **Zynq-7000** and its development tools. Furthermore will **develop** a **deployment** and **testing** of a hardware module, and will be implemented in a custom hardware design of the system. Finally, will develop a **working platform** to allow **software development** in the system.

The main objective with the realization of this research project is to be **a base for a project** to develop hardware and software suited to the needs of a client.

Agradecimientos

En primer lugar agradecer a mi familia el apoyo que siempre me han dado, en especial a mis padres y abuelos, el hecho de que puedan sentirse orgullosos de mi persona y que esperen que yo sea lo que ellos esperan, me ha dado a motivación necesaria para avanzar día a día durante estos años a pesar de mis constantes pérdidas de interés y de esfuerzo.

Me gustaría agradecer a toda la maravillosa gente que he conocido en la titulación, todos los compañeros de clase, que me han ayudado a sentirme integrado. Sobre todo a mis compañeros de prácticas y grandes amigos, Andrés, Javi, Victor, Roberto y Christian, que habéis tenido que aguantar mi peor cara: irresponsabilidad, soberbia y falta de actitud, y habéis tirado de mi cuando lo necesitaba, me habéis escuchado, hablado y habéis hecho que venir a clase, trabajar o estudiar sea todo un placer.

Después querría agradecer el resto de la gente que me ha acompañado durante este largo camino. Los que estuvieron, los que han estado siempre y los que llevan poco tiempo en mí, gracias a vosotros por haberme apoyado cuando peor me he sentido y por ayudarme a avanzar, por darme esa confianza y por seguir creyendo en mí.

Finalmente agradecer a todos los miembros del laboratorio de informática, que me han acogido como uno más entre ellos dándome momentos inolvidables. Un especial agradecimiento a mi tutor Oscar, por tener la enorme paciencia de lidiar conmigo y mi falta de actitud. Y sobre todo por darme una oportunidad, un lugar de trabajo, información, guía y apoyo cuando más perdido he estado y más ayuda he necesitado.

La realización de este proyecto no habría sido posible sin todos y cada uno de vosotros. Muchos días me emociono al pensar la inmensa suerte de estar rodeado de la gente increíble que me rodea, y entre todos habéis podido convencerme que soy capaz de abrazar y llegar a cumplir mis sueños, y cumplir las metas que me proponga.

Este proyecto y esta meta lo he conseguido realizar algo que hace un año lo veía imposible e inalcanzable, sois increíbles, nunca encontraré las ganas de agradecéroslo lo suficiente a todos.

Índice

Tabla de ilustraciones

Índice de tablas

1 Introducción

Este documento se utilizará como herramienta para explicar el proceso de investigación realizado para un proyecto de fin de carrera. Este proyecto intenta explicar el proceso completo necesario para poder añadir un periférico lógico en un dispositivo programable elegido. En este documento se intentará plasmar el trabajo de investigación realizado.

A continuación en este capítulo se realizará una primera toma de contacto con el documento, explicando la motivación para su realización y los objetivos que se quieren alcanzar con su realización. Finalmente se explicarán brevemente los capítulos o apartados por los que está formado el documento, y el contenido de cada uno de ellos.

1.1 Motivación del proyecto

En la actualidad, el **equilibrio entre potencia y consumo** es muy importante en el mundo de los **microprocesadores** para **sistemas móviles y empotrados**, debido a la necesidad de una mayor potencia. Para ello se necesita un sistema de un **tamaño** y **consumo reducido**, y que a su vez tenga la suficiente **potencia**, **capacidades** y **herramientas** que exigen los **desarrolladores** de hardware y software. Debido a esto, las compañías del sector están continuamente buscando **diferentes tecnologías**.

Este proyecto surgió de la curiosidad que despertaba el **Zynq-7000**, el nuevo producto del mayor fabricante de **FPGA, Xilinx**. Este producto trata de aunar **varias tecnologías** novedosas de la computación y de los dispositivos programables, para intentar **extender** el **mercado** de las FPGAs. Este dispositivo se trata de un **SoC** que tiene la **particularidad** de contener una **FPGA** en su interior, y capaz de utilizar todas las tecnologías y herramientas de Xilinx. Todos estos conceptos se explicarán en el apartado de estado del arte del documento.

La **principal motivación** es la de intentar conseguir **implementar** un **periférico lógico personalizado**. Además se tienen **otras motivaciones** por las que se quiere realizar la investigación sobre el Zynq-7000:

- Resulta interesante analizar si esta **nueva tecnología** puede **adaptarse** al **mercado** de dispositivos electrónicos de consumo medio/bajo y sistemas empotrados que requieran una gran potencia.
- Se quiere comprobar si este dispositivo se puede **adaptar** a las **necesidades** del mercado, aprovechando sus características de **dispositivo programable.**
- Se busca conocer si sus herramientas de **desarrollo** son suficientemente **potentes** e **intuitivas** para complacer a los desarrolladores de HW y SW.

1.2 Objetivos del trabajo

El **principal objetivo** que se quieren alcanza con la realización de este proyecto es conocer el funcionamiento de las herramientas y características del dispositivo elegido, realizando una **investigación**, **desarrollo** y **prueba** utilizando una placa de desarrollo.

Con la realización de este trabajo dirigido los **objetivos** que se quieren alcanzar son los siguientes:

- **Investigar y analizar** cómo el **funcionamiento** del hardware del Zynq-7000. Se desean conocer las características técnicas del SoC, debido a que puede ser útil para su utilización. Sin profundizar demasiado en su nivel más puramente físico, debido a que se quiere centrar en la parte de implementación e informática, que en los detalles electrónicos.
- **Conocer y aprender** a utilizar el **entorno de desarrollo de Xilinx** para poder diseñar e implementar lógicas programables, Además se quiere conocer qué otras herramientas se podrían utilizar para ejecutar código en la Zedboard.
- **Conocer el funcionamiento interno de una FPGA**, informándose sobre su funcionamiento y posibilidades, conocer el funcionamiento interno al **implementar una lógica programable**.
- Realizar una **implementación** hardware de un **periférico** con propiedad intelectual conocido como **IP**, para poder ser implantado en la **lógica programable** de una FPGA, como una parte del sistema.
- Crear una **descripción** hardware del **sistema** completo del **Zynq-7000** adaptada a nuestras necesidades. A esta descripción se tiene que **añadir** y **configurar** el **periférico** creado, realizando la **intercomunicación** internamente mediante un **bus**.
- **Probar** el correcto **funcionamiento** del **periférico** mediante la ejecución de **pruebas** sobre **varios niveles**:
    - Comprobar su correcto **funcionamiento hardware**, utilizando **benchmarks** sobre el periférico en un **simulador**, realizando **estimulación de señales**.
    - Verificar que el periférico se ha **añadido correctamente** en el **sistema** completo. Revisar el correcto **funcionamiento** de las **interconexiones**, comprobar que se puede **acceder** a sus **registros** al descargar la descripción HW del sistema completo en la placa.

1.3 Contenido de la memoria

En este documento tratará de explicar y plasmar todo el trabajo de investigación que se ha realizado. Para ello en primer lugar se realizará una descripción sobre el sector tecnológico el cual se centra el proyecto: **FPGAs**, **arquitecturas de microprocesadores** y **System on Chip.** Explicando sus orígenes, su evolución y la situación actual en la que se encuentran con sus tecnologías más novedosas.

Posteriormente se **explicarán** las **herramientas de desarrollo** escogidas para la implementación hardware y software. El entorno escogido se trata del entorno **ISE** del fabricante del chip que utilizaremos, y se explicará el **funcionamiento** de sus **principales programas** por los que se compone **brevemente**.

Posteriormente se explicará el **análisis** del proyecto realizado, realizando un análisis sobre las **capacidades** y **restricciones generales** del proyecto, sobre el entorno de desarrollo que se ha utilizado, un análisis sobre el dispositivo que se ha escogido para realizar la investigación y finalmente se redactará un ligero catálogo de requisitos sobre los diseños implementados.

Después se explicará el **diseño** de la implementación hardware y software realizado por nosotros, por lo que se explicará el **funcionamiento** del **código** diseñado, y se explicarán las **decisiones de diseño** realizadas.

Una vez explicado el **diseño**, se detallará la instalación del entorno de desarrollo y los pasos que han sido necesarios para la **implementación** del todo el código diseñado, desde la implementación del **diseño del microprocesador**, la implementación del **periférico** y su **importación**, y su implantación en la placa. Después se mostrará el resultado de la **ejecución** todas las pruebas **realizadas**.

Finalmente se realizarán unas **conclusiones** sobre el trabajo realizado e ideas para posibles **trabajos futuros** que pueden surgir a partir de este.

Las **secciones** contenidas en este documento son las que se explican a continuación:

- **Estado del arte:** En esta sección se dará una información general sobre el entorno que está dirigido este trabajo. Particularmente se explicará de qué trata una **FPGA**, su funcionamiento y su uso habitual, y sus arquitecturas de procesadores lógicos denominados **Soft Processors**. Posteriormente se explicará muy brevemente el funcionamiento de un microprocesador, explicando un poco más detenidamente el funcionamiento de las entradas salidas, y se enunciarán y explicaran brevemente las principales **arquitecturas de microprocesadores**, en particular la que utiliza el Zynq-7000, comentando sus principales **características** y **ventajas**.

- **Análisis:** En este apartado se **analizarán** las **características** y **especificaciones** del producto que se quiere probar, la placa Zedboard, así como describir detalladamente el entorno dónde se van a realizar e implementar las pruebas en la placa.
- **Diseño**: En este apartado se comentará brevemente el **diseño hardware** que se ha realizado para **implantarlo** en la placa Zedboard. También se describirá muy brevemente el **código** que se ha utilizado para probar el funcionamiento de la placa de desarrollo.
- **Implementación**: En este apartado se comentará el proceso que se ha realizado para poder probar la placa, explicando tanto la **instalación** del entorno de desarrollo como la **implementación** de las **pruebas** y finalmente la **ejecución** en la placa de varias maneras. En este apartado se utilizarán **capturas de pantalla**, para ir describiendo los pasos qué se han realizado para poner en funcionamiento la placa y conseguir ejecutar las pruebas diseñadas.
- **Conclusiones**: En este apartado se comentará las impresiones y conclusiones obtenidas con la realización de la toma de contacto. Así mismo se expondrán los principales problemas que se han encontrado.

Adicionalmente se ha añadido varios apartados donde se explicarán los términos, **abreviaturas** que se han utilizado y la **bibliografía** básica utilizada.

2 Estado del Arte

En este apartado se **explicarán** brevemente las **tecnologías** y **productos** que están relacionadas con el SoC Zynq-7000. También se realizará una breve introducción de la **historia** de cada tecnología, explicando su **pasado** y **presente** haciendo énfasis en el estado actual de cada tecnología.

2.1 Introducción a los dispositivos programables

Desde el **transistor**, al **circuito integrado** o **Chip**, pasando por el **ASIC**, el siguiente avance de la industria electrónica y de los semiconductores fueron los dispositivos de **lógica programable**.

Al inicio de los años 70 comenzaron a crearse los **primeros dispositivos** con **lógica programable**, conocidos como PLD (*Programmable Logic Device*). Estos dispositivos al contrario de un sistema de puertas lógicas, no tenían una función programada en su fabricación.

Posteriormente estos dispositivos programables fueron evolucionando a **otros dispositivos** de menor tamaño y mayor velocidad, y después se realizaron de mayores tamaños formados por un grupo de PALs denominados CPLDs, capaces de remplazar circuitos de gran tamaño equivalentes a miles de puertas lógicas. Estos dispositivos necesitaban una **programación realizada por el fabricante**, y estaban muy orientados a los **circuitos digitales** y **puertas lógicas**.

Por otro lado en los primeros años de la década de los 80, los semiconductores llegaron a un punto en el que podían ser integrados en **un solo chip**, cada vez más **específicos** y **complejos**. Las empresas proveedoras de chips eran **incapaces** de fabricar chips para todos sus clientes, y el diseñador no tenía los medios para fabricar circuitos integrados modernos adaptados para su sistema.

Para intentar resolver este problema, nacieron los **ASIC** o circuitos integrado para aplicaciones específicas, chips diseñados para desempeñar un **propósito específico**. Para la fabricación, el diseñador realizaba un diseño válido del sistema usando **herramientas del fabricante**, que posteriormente enviaba a éste para que fabricara los circuitos integrados.

A mediados los años 80, se **cruzaron los caminos** entre las tecnologías ASIC y PLD. Mientras que la industria de los dispositivos programables se encargaba de fabricar PALs y sus evoluciones como los dispositivos GALs o CPLDs, surgió otra corriente de desarrollo distinta basada en la **tecnología de matriz de puertas** (*Gate Array*).

En 1984 Ross Freeman y Bernard Vonderschmitt, cofundadores de **Xilinx**, crearon el primer dispositivo programable con la tecnología basada en la matriz de puertas, este dispositivo se denominó como **FPGA** (*Field Programmable Gate Array*). La idea era la de crear un dispositivo que se pudiera **programar** en función de los requisitos de la aplicación.

Como indica su nombre, *Field Programable,* a diferencia de los dispositivos de la época, las FPGAs podían ser **programadas por el cliente** sin ser necesario ser programados o reprogramados desde fábrica. Además este dispositivo podía almacenar una **lógica más compleja**, pudiendo programar tanto puertas lógicas, circuitos digitales como los anteriores dispositivos, o incluso sistemas computacionalmente más complejos como **microprocesadores**.

Gracias a estas dos ventajas, permitía muchas cosas muy interesantes para la computación tanto para fabricantes como para diseñadores, tanto **software** como **hardware**.

Resumiendo, las principales **ventajas** a destacar de las FPGAs son las siguientes:

- Posibilidad de la **prueba y simulación** de diseños antes de la implementación.
- Creación de **familias** de productos de una forma **barata y sencilla.**
- Posibilidad de **adaptación a los sistemas** incluso a un nivel software a un coste más bajo al no ser dispositivos específicos.
- **Reconfiguración** y **programación** de una manera más **eficiente y económica**.
- Capacidad de **implementar** microprocesadores, memoria y otros dispositivos implementados mediante **síntesis de puertas lógicas**.

Dependiendo del tamaño de los dispositivos programables y de su potencia es posible implementar *Soft Processors* más complejos y con más recursos, incluso implementar **varios** *Soft Processors* en un mismo dispositivo, aunque provoca dificultades por la compartición de los recursos.

Las principales arquitecturas *de Soft Processors* son las siguientes [13]:

- **Microblaze**: es la arquitectura propuesta por el fabricante de FPGAs **Xilinx** basado en una arquitectura RISC. Es capaz de ejecutar la mayoría de las instrucciones en un solo ciclo.
- **Nios II:** es la arquitectura propuesta por el otro principal fabricante de FPGAs **Altera**. Esta es una arquitectura de 32 bits, y también se trata de una arquitectura RISC.
- **LEON 4**: es una arquitectura desarrollada por la **ESA** con el fin de implementar un procesador software de alto rendimiento para utilizarlo en proyectos espaciales.
- **Open Risc**: son una serie de arquitecturas RISC de **código libre**.

Las **principales** empresas **fabricantes** de FPGAs son **Xilinx** (La creadora de la tecnología) y **Altera**, su principal competidora. En la actualidad entre las dos empresas tienen más del **80%** del mercado de la **fabricación** de FPGAs, con herramientas para la programación de sus dispositivos para SSOO Windows y Linux.

En la **actualidad** las FPGA han ido avanzando en su escala de integración, para poder introducir **más** cantidad de semiconductores en el mismo o menos espacio físico. Además de estas mejoras, también se han producido **mejoras** a más alto nivel, como la **mejora** de sus **herramientas de desarrollo** del hardware lógico y software, y la inclusión de **nuevas tecnologías** como *"**partial reconfiguration**"*.

Particularmente la tecnología de *"partial reconfiguration"*, permite la **reconfiguración** o **reprogramación** de un **módulo** o **sector** de la FPGA mientras el resto de la lógica sigue corriendo con normalidad.

Las FPGA se han utilizado comúnmente para varios sectores de la industria electrónica, comúnmente en el sector de los sistemas empotrados. En la actualidad se han ampliado los sectores en los que se utiliza debido a la mejora de rendimiento.

Los principales sectores de la industria y usos de las FPGA en la actualidad son los siguientes:

- **Industria Aeroespacial** y **Defensa**: Espacio, misiles y munición, soluciones de seguridad.
- **Realización de prototipos para ASIC:** Realización, simulación y prueba antes de desarrollar un dispositivo ASIC.
- **Multimedia**: reproducción de video en alta resolución en local o en red, imagen industrial, radio, procesamiento de señales digitales, tecnologías portátiles.
- **Automoción**: procesamiento de imágenes, conectividad y red de vehículos.
- **Servidores de datos y Redes**: Video en tiempo real, controladores para *routers* y *switchs*, balanceo de carga de servidores y seguridad.
- **Medicina**: Ultrasonidos, Rayos-X, sistemas quirúrgicos y resonancias.

2.2 Arquitectura ARM

La arquitectura **ARM** es una arquitectura de microprocesadores creada por la empresa *Acorn Machines* para sus PC. Como su acrónimo indica *Advanced Risc Machine*, se trata de una arquitectura de microprocesadores basada en la tecnología **RISC**.

La arquitectura está basada en *MOS 6502*, adaptándola para conseguir una **mayor potencia**, con el mismo rendimiento de entrada/salida y con un conjunto de **instrucciones de 32 bits**.

ARM distribuye varios tipos de microprocesadores dependiendo del mercado al que va orientado. Además del hardware que fabrica, la empresa provee de un **entorno de desarrollo** (SDK), un **compilador** y herramientas de **depuración**. Los diferentes modelos de ARM según el mercado al que van dirigido son los siguientes [12]:

- **Series Cortex-A**: Se trata de microprocesadores de **alto rendimiento** dirigido a sistemas operativos abiertos. Suelen ser microprocesadores con varios núcleos de alta frecuencia con una unidad de coma flotante incluida. Se suelen encontrar en teléfonos móviles, *tablets* y videoconsolas de **última generación**.
- **Series Cortex-R**: Se trata de una gama de microprocesadores más orientado a los **sistemas empotrados** y **sistemas de tiempo real**. Tienen un consumo bajo y un buen rendimiento.
- **Series Cortex-M**: Se trata de una gama de microprocesadores de **tamaño reducido** y es la gama que tiene el **menor consumo** de todas. Están orientados para su uso en micro controladoras por norma general. El modelo **Cortex-M1** se utiliza para implementarlo en **FPGAs** como *soft processor*.
- **Series Classic**: Son su gama de procesadores **más extendidos** en el mercado con más de 20 billones de dispositivos en el mercado, siendo una solución **económica**. Los modelos de esta familia son los procesadores ARM9, ARM7 y ARM11 entre otros.
- **Series Especiales**: Otras arquitecturas con alguna funcionalidad o particularidad **especial,** como por ejemplo el modelo *SecurCode*, el cual contiene varios módulos específicos para seguridad.

En la actualidad los microprocesadores ARM son desarrollados por la empresa ARM Holdings, siendo los microprocesadores **más extendidos** en todo tipo de sistemas empotrados. Los procesadores ARM tienen en la actualidad más del **90% del mercado** de procesadores de 32 bits de tecnología RISC. Está **muy extendido** en muchos mercados de la electrónica de consumo debido a su buena relación **potencia/consumo**, destacando **videoconsolas**, **telefonía móvil**, ***tablets***, periféricos y controladoras de dispositivos informáticos.

Los principales alternativa a las arquitecturas ARM en el mercado de los microprocesadores de bajo consumo se tratan de la arquitectura MIPS y versiones antiguas de la arquitectura x86.

La arquitectura **MIPS** es una **arquitectura** basada en la arquitectura **RISC** desarrollado por la empresa MIPS Computer Systems. Las implementaciones de MIPS se han utilizado usualmente en **sistemas empotrados** tales como Windows CE, *routers*, videoconsolas (PSP, PS2). A mediados de los 90 más del 30% de los microprocesadores eran procesadores con una implementación Mips, actualmente los procesadores MIPS han sido **sustituidos por la arquitectura ARM** convirtiéndose esta última en la líder indiscutible del mercado.

Por otro lado, la arquitectura **x86** desarrollada por **Intel** introduciéndola la primera vez con el procesador 8086. La arquitectura está basada en un juego de **instrucciones CISC**, y tiene la peculiaridad de ser una arquitectura parcial o totalmente **retrocompatible** con sus diferentes evoluciones, denominado como una **arquitectura no limpia**. En la **actualidad** el uso de esta arquitectura ha sido casi **descartado** de este mercado, al **orientarse** sus evoluciones hacia el **mercado** de los computadores personales o **PC**, con un **consumo** y un **rendimiento** muy **superior**.

2.3 System On Chip

Se denomina un **System on Chip** o System on a Chip, definidos por sus siglas **SoC**, a un **circuito integrado** que integra todos los componentes de un computador u otro dispositivo electrónico en **un sólo Chip**. El diseño de estos sistemas puede estar basado en señales analógicas, digitales o incluso mixtas, y pueden llevar a menudo sistemas de radio-frecuencia como *wifi*, *bluetooth* o redes móviles.

La **evolución** de las tecnologías de los semiconductores y particularmente el aumento de la escala de integración permite la inclusión de **más cantidad** de semiconductores en un **menor espacio físico**.

La evolución ha hecho posible el hecho de **integrar todos** los **componentes** un computador, comúnmente separados, en un solo chip. Esto no solo provoca una **reducción** de la **complejidad** del sistema, el **coste** y el **consumo de energía**, sino que además **reduce** en gran medida el **espacio físico**, haciendo posible **integrar** un computador de **alto rendimiento** en dispositivos con un **espacio reducido**, donde usualmente se usaban sistemas empotrados de una potencia inferior.

Este hardware está acompañado por un **software** específico que maneja todos los elementos importantes del sistema el **microprocesador**, **microcontrolador**, **DSP**, lógica programable y todos los periféricos.

Este software comúnmente se diseña al mismo tiempo que el diseño hardware para estar **adaptado** y **optimizado** para este **SoC**, y se suele seguir un **proceso de desarrollo** similar a la implementación de **software específico** en ASIC.

Comúnmente los SoC contienen procesadores lo suficiente **potentes** para poder cargar las versiones de escritorio distintos **sistemas operativos** como Windows o Linux, o sus respectivas versiones móviles, aunque normalmente necesitan alguna memoria externa (RAM, Flash) para que sean totalmente operativos.

La mayoría de los SoC están desarrollados por **módulos de hardware** previamente probados y testeados. Normalmente estos módulos hardware se adaptan a la familia de protocolos a la que pertenecen, por ejemplo, el caso de las interfaces **HDMI** o **USB**.

Normalmente en un SoC podemos encontrar los siguientes componentes integrados:

- Uno o varios núcleos **microcontroladores**, **microprocesadores** o **DSPs**.
- Bloques de memoria **ROM**, **RAM**, **EEPROM** o **Flash**.
- Fuentes de reloj como **osciladores** o **PLL**.
- Periféricos incluyendo contadores y **relojes** de tiempo real.
- Interfaces externas para periféricos como **USB**, **HDMI**, *Firewire*, ***Ethernet***, UART o interfaz serial.
- Reguladores de voltaje y de potencia.
- Comunicaciones radio-frecuencia: sistemas **WiFi**, **Bluetooth** o **redes móviles**.
- Unidades de Procesamiento de Gráficos (**GPU**) o unidades de coma flotante (**FPU**).
- **FPGAS** y dispositivos programables.

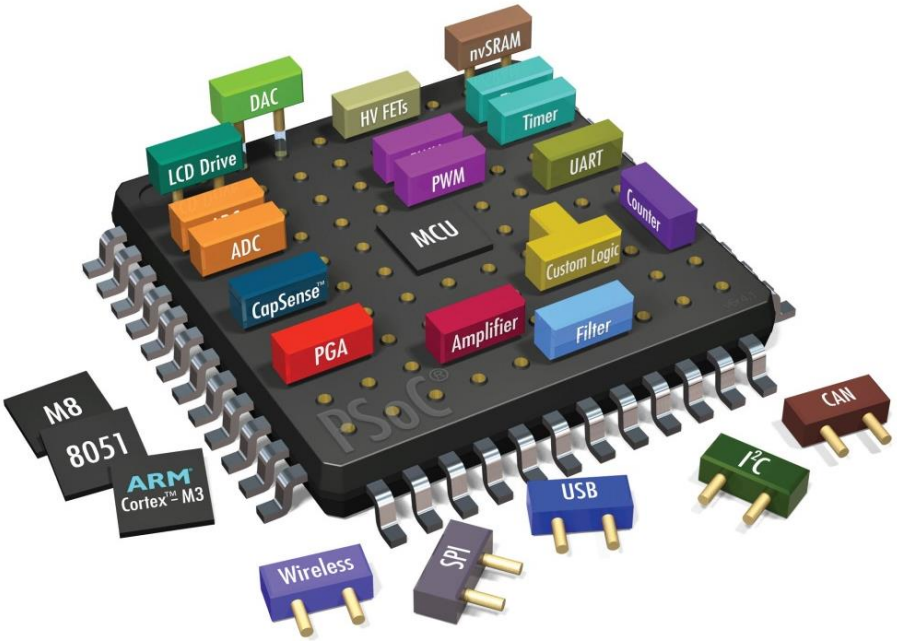


Ilustración 1: Imagen explicativa SoC

Por norma general la mayoría de **SoC** que contienen un módulo con microprocesadores en su interior, llevan microprocesadores pertenecientes a la **arquitectura ARM**, debido a que es la arquitectura que más se **adapta** a sus necesidades, debido a su buena relación **potencia / consumo**.

A continuación se explicará brevemente el funcionamiento del Zynq-7000, para dar información de su funcionamiento y composición.

2.3.1 Descripción Zynq-7000

La gama **Zynq-7000** se trata de un SoC único que contiene la particularidad de además de un **microprocesador** ARM, contiene una **FPGA**. El chip está formado por **dos partes** interconectadas que se pueden distinguir, una parte **física** y por otro lado la **lógica programable**. En la parte física podemos encontrar el **sistema de procesamiento** o **PS** (*Processing System*) por un lado y las **entradas/salida multiplexadas** por otro. En la parte **lógica programable** o **PL** (*Programmable Logic*) es utilizada para adaptarla según las necesidades.

En el siguiente diagrama se puede ver la composición del SoC utilizado y apreciar sus diferentes secciones:

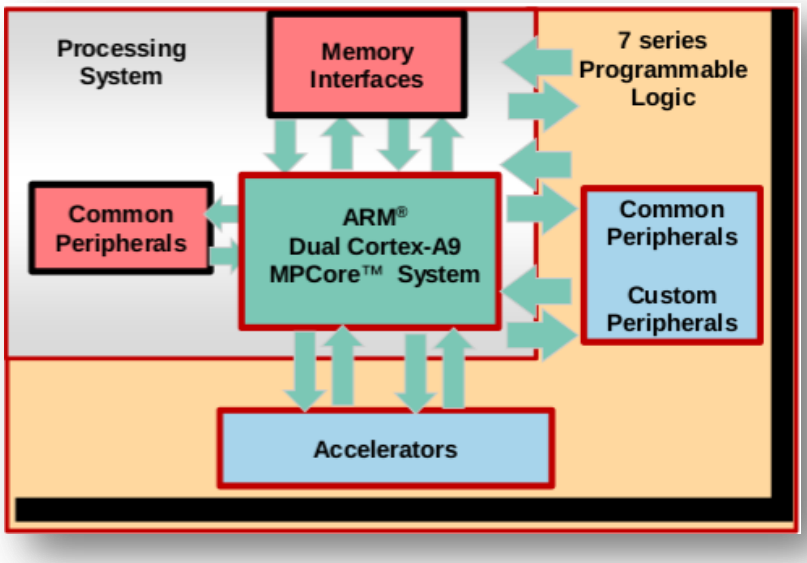


Ilustración 2: Diagrama de arquitectura Zynq

Como se puede observar en la ilustración, la parte del **sistema de procesamiento** está formado por el **microprocesador**, la interfaz para la **memoria RAM** y parte de las interfaces para periféricos. Por otro lado la **lógica programable**, puede llevar algún **acelerador** necesario para la ejecución de código, como **softprocessors**, **memoria**, **FPUs** o cualquier dispositivo extra que sea capaz de sustituir la lógica programable.

La gestión de las señales de **entrada/salida** de las interfaces de periféricos **comunes**, están **repartidas** entre la lógica programable y la parte física, **separándo** la gestión de la entrada/salida entre las secciones PS (*Processing System*) y PL (*Programmable Logic*) del chip.

Para controlar las interfaces de los periféricos, en la parte **PS** tenemos un sistema **multiplexado** conocido como **MIO** (*Multiplexed Input/Output*), que controla las entradas/salidas de algunos dispositivos de un uso más habitual y **general**.

Por otro lado en la parte **PL** se controlarán las interfaces de otros periféricos que por norma general tendrán un **menor uso** o una mayor carga computacional. Adicionalmente en el caso de los las interfaces de Audio o HDMI y de algunos Pmods, la lógica programable también se encargará de implementar el **CODEC de audio**, o el **transmisor HDMI** y **controladores intermedios** necesarios. Resulta muy útil que los dispositivos menos habituales se puedan **desactivar**, y **aprovechar** esa parte de la lógica programable para otras utilidades como aceleradores.

La ubicación de cada interfaz que controla el SoC se puede apreciar en el siguiente diagrama:

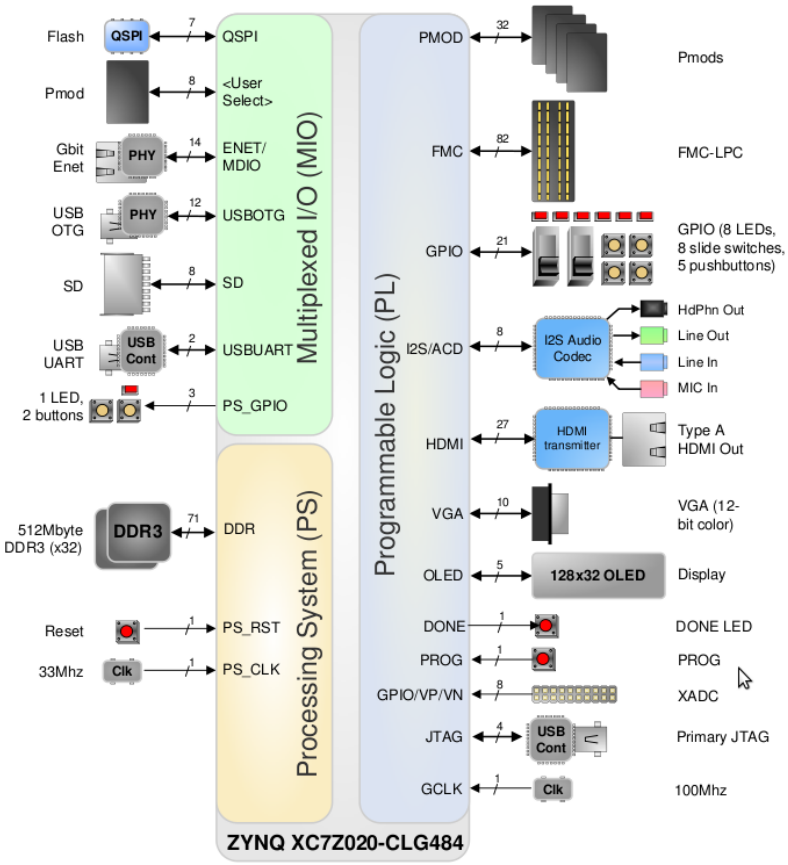


Ilustración 3: Diagrama de periféricos en Zynq

El SoC siempre cargará su **configuración inicial** al encenderse desde la **memoria Flash**, cargando el contenido de su interior. Auxiliarmente el SoC, puede cargar la configuración inicial desde una tarjeta SD, permitiendo configuraciones de un mayor tamaño, pudiendo **cargar un sistema operativo** en su totalidad.

Finalmente es importante indicar que para poder aprovechar el SoC, existen varias herramientas para la implementación de diseños hardware y software.

2.3.2 Interconexión SoC

Los SoC tienen una interconexión mediante **buses** para lograr la **intercomunicación** entre sus **bloques** por los que está compuesto.

En sus **inicios**, los SoC tenían sistema de **intercomunicación simple** similar a la que tenían los sistemas de microprocesadores en sus inicios. Este sistema se componía simplemente de un **único bus** del sistema que **interconectaba todos los bloques** por los que estaba compuesto el SoC.

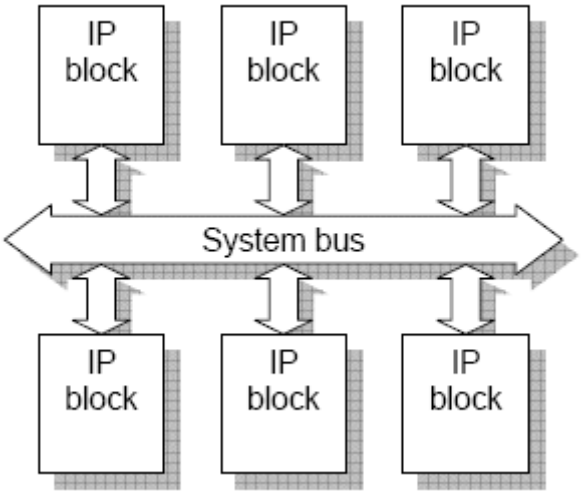


Ilustración 4: Interconexión básica SoC

El sistema era **sencillo** y **funcional** a **baja escala**, debido a que este sistema tiene una **mala escalabilidad**. Debido a esto y a las **necesidades** de SoC más **complejos**, con un **mayor número de bloques**, **mejor rendimiento**, los SoC no se pueden construir ante un solo bus del sistema.

Ante esta situación, se desarrolló una **jerarquía de buses** más compleja, con **protocolos** para los **buses** más elaborados y con ***bridges*** entre ellos. De esta manera **un bloque** puede estar comunicado por **varios buses** al mismo tiempo, y la comunicación con estos buses tiene que ser verificada minuciosamente.

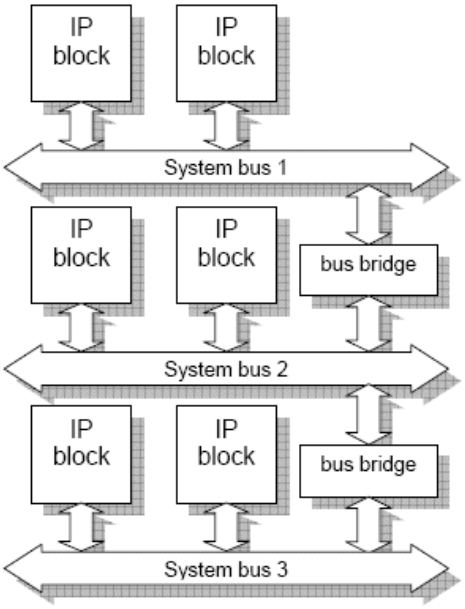


Ilustración 5: Interconexión con jerarquía de buses en SoC

En los **protocolos** de los buses para la interconexión dentro de los SoC, se implantaron los protocolos **diseñados** en un primer lugar para los **buses para microcontroladores avanzados** y desarrollado por la empresa creadora de la arquitectura más utilizada en los procesadores de los SoC, **ARM**. Este conjunto de protocolos se denomina **especificación AMBA**, y fue incrementándose su utilización en SoC y en ASIC, actualmente es el **estándar** para los **procesadores empotrados** de 32bits.

La especificación del estándar va por su cuarta versión **AMBA4**, esta especificación está compuesta por los siguientes protocoles para la interconexión con buses:

- **AXI Coherency Extensions** (ACE): Utilizado en los más modernos procesadores de la gama Cortex–A, en los que se engloban los procesadores **Cortex-A7** y **Cortex-A15**.
- **AXI Coherency Extensions Lite:** Versión reducida y más sencilla del protocolo **ACE**.
- **AXI 4.**
- **AXI 4 Lite:** Versión reducida y más sencia del protocolo AXI 4.
- **AXI 4 Stream:** Versión de AXI 4 diseñado para enviar una gran densidad de datos a través del bus.

2.4 Herramientas de Desarrollo Zynq-7000

El diseño e implementación para el **SoC** Zynq 7000 se centra en dos tipos bastantes diferenciados, el **desarrollo hardware** y el **desarrollo software**.

Por un lado, el **desarrollo hardware** se encarga de **diseñar, comprobar, simular e implementar** todos los posibles **periféricos personalizados** en la parte programable del Zynq, es decir introducidos en su sector de lógica programable o PL, correspondiente a la FPGA de su interior.

El desarrollador hardware busca **configurar** el Zynq, modificando su hardware **añadiendo periféricos** y **gestionar** su **entrada/salida**. Para poder realizar esto, el desarrollador hardware puede **añadir** estos **periféricos** como **bloques**, ya sea añadiendo ofertas **existentes** en el mercado o periféricos propios con una **lógica personalizada**, comúnmente estos bloques de periféricos se denominan ***IP cores*** o ***IP Blocks*** cuando tienen propiedad intelectual.

Para que el desarrollador hardware pueda **interconectar** el sistema de procesamiento o **PS,** con la lógica programable o **PL,** necesita un medio de comunicación entre ambos. Esta interconexión mediante enlaces directo o **FSL** transmitiendo información mediante un **BUS** interno, y los registros de los periféricos.

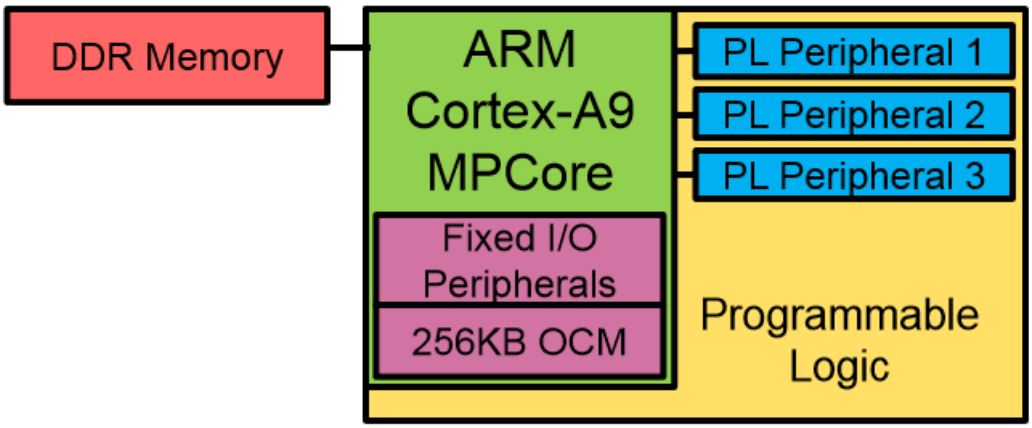


Ilustración 6: Visión Desarrollador HW del Zynq

Por otro lado el **desarrollo software** trata de poder diseñar e implementar cualquier tipo de código, incluyendo SSOO, capaces de utilizar todo del hardware del Zynq, incluyendo los periféricos creados en la PL.

Para poder **interactuar** con los bloques de periféricos creados por los **desarrolladores hardware**, el PS controla los bloques de la PL mediante las interfaces de sus registros exportados, y permite **acceder** al desarrollador software a esos **registros** mediante una arquitectura de **direcciones y mapeado estándar**, similar a todos los sistemas de propósito general. Es decir, el desarrollador software podrá acceder a los registros de esos bloques accediendo a un rango de **direcciones de memoria**.

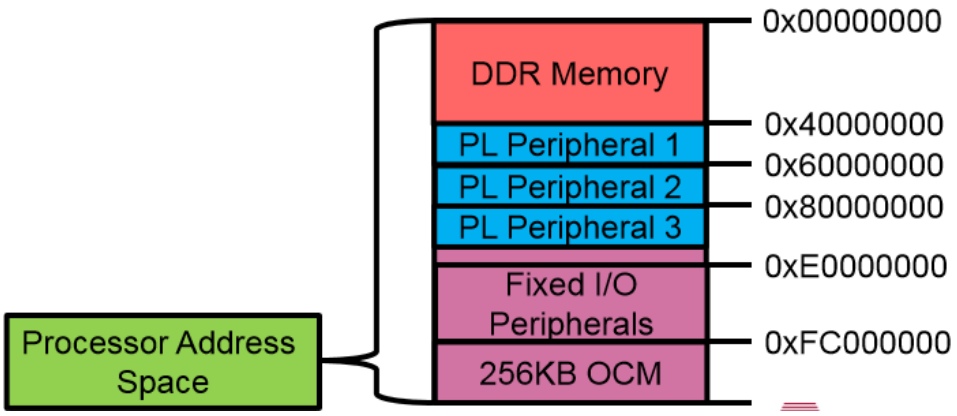


Ilustración 7: Visión Desarrollador SW del Zynq

Para este desarrollo hardware y software para el Zynq existen muchas **herramientas** para su realización. En general existen múltiples herramientas para sacar partido al SoC, por norma general estas herramientas se agrupan en **suites de desarrollo**. Las principales suites de desarrollo son las siguientes:

- **MATLAB/Simulink**: Herramienta de desarrollo creada por la empresa Mathworks, especializada en **cálculo matemático**. En su mayoría se utiliza el programa Simulink que tiene completa integración con Matlab, es una **herramienta de desarrollo multilenguaje** para el **diseño y simulación** de **modelos** o **sistemas** con un alto grado de elementos físicos. Su más interesante característica es la de convertir esos modelos en IP Cores mediante el uso de otra herramienta.
- **ARM DS**-5: El *ARM Develelopment Studio* es una herramienta de desarrollo creada por la empresa creadora de la arquitectura del procesador del SoC, **ARM**. Se trata de una herramienta que permite **desarrollar software** robusto y optimizado para los procesadores ARM. La suite contiene **varias herramientas** para poder sacar todo el provecho al procesador, la cadena de herramientas engloba el **compilador de C/C++** para ARM, un **debugger** para **Linux/Android/RTOS**, el **analizador de rendimiento** y **simulador** de modelos y sistemas reales *ARM Streamline*, todas correctamente empaquetadas en un entorno de desarrollo (**IDE**) basado en Eclipse.

- **Xilinx ISE Design Suite:** Se trata de la suite principal que provee el fabricante de la FPGA del SoC, Xilinx. Está formado por una **serie de herramienta**s para intentar cubrir todas las necesidades del desarrollador **hardware y software**. La suite ISE de Xilinx se compone en la última versión de los siguientes programas orientados al desarrollo para el Zynq:
    - **ISE:** Es una herramienta de desarrollo orientada al **diseño hardware** realizados con lenguajes **HDL**, en particular **VHDL** y **Verilog**. Esta herramienta principalmente permite **sintetizar** e **implementar** de los diseños escritos en HDL creados. También se puede utilizar para realizar **análisis de tiempos**, examinar **diagramas RTL** y utilizar su simulador **ISIM** para probar modelos mediante estimulación de señales. También contiene un programa que se permite la creación de **IP Cores** mediante el uso de plantillas, llamado **Core Generator IP**.
    - **XPS:** Es una herramienta para poder **diseñar hardware** para sistemas empotrados. En general permite realizar un cometido parecido a la herramienta ISE mediante tutoriales, con una interfaz gráfica y de una forma más **sencilla** e **intuitiva**. Una vez creados los diseños, permite **crear** y **añadir periféricos** e **IP Cores** , y gestionar los puertos y buses de estos IP, y la asignación de **direcciones de memoria.**
    - **XILINX SDK**: El Xilinx Software Development Kit es una herramienta que permite el **desarrollo** de **software** para dispositivos programables. La herramienta permite el **diseño**, **compilación**, **debug** y **ejecución** de código escrito en **C/C++** para ejecutarlo directamente sobre el hardware o en un sistema operativo Linux.
    - **PLAN AHEAD**: Permite **gestionar** los **proyectos** para la implementación de un diseño conjunto de hardware y software. Permite **comprobar** y **analizar** de que todos los elementos del sistema funcionan correctamente en su conjunto, permitiendo **importar proyectos desde XPS o ISE**.
    - **ISIM**: Se trata del **simulador** para ISE, que permite realizar **simulaciones** sobre los diseños hardware realizados en ISE. El programa puede comprobar los modelos utilizando **estimulación de señales**, pudiendo observar el comportamiento del módulo cuando llegan diferentes señales.
- **Xilinx Vivado-HLS** (Auto ESL): Se trata de una herramienta que es una **evolución** de la suite de desarrollo ISE de **Xilinx**, enfocada a un **diseño de alto nivel de síntesis.** Esta herramienta tiene la novedad de utilizar **lenguajes de alto nivel** de síntesis o **HLL** como evolución de los lenguajes HDL. La herramienta además aumenta en gran medida la **velocidad** a la hora de realizar a la hora de realizar las **simulaciones**, la **implementación** del modelo, realizar ***place-and-route***, y la generación y comprobación de IP cores. Además mejora la generación e importación de IP cores, pudiendo hacer **IP** de **alto nivel de síntesis** y dando más facilidades para realizar ***partial reconfiguration*.**

3 Análisis del sistema

En este apartado se va a definir el análisis del código y de las pruebas que se han diseñado, así como enunciar y explicar el entorno operacional escogido. Este apartado se ha realizado basándose en una de las metodologías propuestas por la ESA, la metodología **ESA PSS-05 Lite** pensada para proyectos de tamaño reducido. Para adaptarse a las necesidades de la documentación de este proyecto, se han **eliminado** o **modificado** algunas secciones.

En primer lugar se realizará una descripción general del proyecto, donde se enunciarán brevemente las **capacidades** y **restricciones** generales del proyecto en general, y particularmente de los diseños y pruebas que se han realizado.

Posteriormente se detallará el **entorno operacional** que se ha escogido para la realización del proyecto. En este apartado se explicará tanto el **entorno de desarrollo** como el **entorno de prueba e implementación**, enumerando y explicando los diferentes **componentes HW** y **SW** por los que están formados.

Finalmente se realizará un escueto catálogo de **requisitos** de usuario, ampliando y explicando las restricciones y capacidades a las que se tiene que ceñir el desarrollo que se va a realizar.

3.1 Descripción general

En este apartado se realizará una descripción general del **sistema** que se ha realizado, realizando una **breve descripción** del sistema explicando las capacidades y restricciones generales que se detallarán posteriormente en el catálogo de requisitos.

3.1.1 Capacidades General

A continuación explicaremos cuales son los **principales objetivos** y **características** se tienen de cumplir con el diseño realizado. La lista de capacidades que tienen de cumplir tanto el diseño del periférico como las pruebas es el siguiente:

- El periférico implementado tiene que ser capaz de realizar una **suma** de **16 bit** y almacenar el resultado en un registro.
- El periférico tiene que ser capaz de poder ser implementado como un **IP core** o **IP Block**.
- El **diseño hardware completo** del Zynq tiene que ser capaz de poder **importar el periférico**, interconectarlo con el sistema de procesamiento y asignarle direcciones de memoria.

- Se tiene que poder **acceder** a los **registros** vía software del periférico accediendo al **rango de memoria** asignado para este periférico.
- Las pruebas de **simulación**, tienen que comprobar el correcto funcionamiento del hardware del sumador mediante **estimulación de señales**, comprobando que realiza la suma correctamente.
- Basándose en la descripción del hardware realizada, se creará una plataforma de soporte o **BSP** para permitir la ejecución de código **directamente** sobre el hardware.
- Las **pruebas software** del sistema comprobarán que se puede **escribir y leer** de los registros del periférico y que se realiza correctamente la suma, pudiendo obtener un resultado por consola.

3.1.2 Restricciones generales

En este apartado se nombrarán las **restricciones generales** que tiene que cumplir el código desarrollado. La lista de restricciones para el desarrollo del código son las siguientes:

- El desarrollo del código se realizará utilizando como **herramienta de desarrollo** la última versión de la suite de desarrollo de **Xilinx ISE** versión 14.3
- El diseño hardware del periférico se tiene que realizar en los lenguajes de descripción de hardware **VHDL** o **Verilog**.
- Las **pruebas de simulación** sobre el hardware se tendrán que implementar como **testbench** escritos en lenguaje **VHDL**.
- La interconexión entre el periférico y el sistema se realizará utilizando un bus del sistema, utilizando el protocolo AXI4LITE.
- Todo el código **software** encargado de interactuar con el sistema y el periférico estarán escritos en lenguaje **C/C++.**
- La **descripción hardware del sistema** tiene que ser exportado como un ***bitstream*** para la programación en la placa y el desarrollo de software asociado.
- La compilación del software se tiene que realizar con una herramienta de **compilación cruzada**, optimizando el código para el hardware específico implementado.
- La placa de desarrollo tiene que contener un SoC programable de la gama **Zynq -7000** de Xilinx.
- El **equipo de desarrollo** tiene que disponer al menos de una interfaz **USB** para la interconexión con la placa con los **puertos serial y JTAG**.
- La programación de la placa se realizará mediante el puerto **JTAG** o mediante la programación de la memoria **Flash**.

3.2 Entorno Operacional

Para la realización de este proyecto, se tienen **muchas posibilidades** a la hora de escoger el entorno de trabajo, desde el hardware y software del entorno desarrollo como la placa con el SoC para la prueba e implementación.

Para la elección de la plataforma de trabajo se ha intentado **minimizar el coste económico** para el desarrollo del proyecto, por lo que se han buscado **opciones asequibles** o a mano para la elección del hardware y software a utilizar, aptas para un **trabajo de investigación**.

En la elección de la **plataforma hardware**, para la placa de desarrollo se ha buscado una **placa de desarrollo** económica que incluya un **SoC programable Zynq-7000**. Por lo que se ha optado por una placa de desarrollo de bajo coste, orientada para la educación e investigación, la placa de desarrollo **Zedboard** fabricada por **Digilent**.

Por otro lado, el equipo de desarrollo tiene que ser un PC con todas las interfaces necesarias para la comunicación con la placa y una **potencia de procesamiento media o alta**. A pesar de que la potencia de procesamiento no es determinante para el resultado final, el tiempo necesario para exportar la generación del fichero *bitstream* varía en gran medida dependiendo de su potencia. Debido a esto se ha utilizado un PC con un procesador de última generación i7–*Sandy Brigde*, y con memoria suficiente para poder utilizar las herramientas de desarrollo.

A la hora de la **elección del software** se ha buscado la utilización de herramientas económicas o gratuitas. El equipo de desarrollo utilizará la última versión de la suite de desarrollo **Xilinx ISE**, que contiene todas las herramientas de desarrollo necesarias para la realización. El sistema utilizará un **SSOO GNU/Linux** y diversas herramientas y drivers de software libre para la puesta a punto del entorno de desarrollo.

A continuación se explicará más **detalladamente** la composición y el funcionamiento del hardware y el software tanto de la placa como del entorno de desarrollo.

3.2.1 Zedboard

La **Zedboard** es una placa de desarrollo de bajo coste que contiene en su interior el SoC Zynq-7000. Esta placa está fabricada por los fabricantes de dispositivos electrónicos **Avnet** y **Digilent**, existen diferentes versiones de la placa: Avnet fabrica una placa para uso profesional y Digilent para un uso académico.

Esta placa de **bajo coste** (alrededor de 300 euros) se utiliza en su mayoría para ser un **punto de partida** para diseñadores de hardware y de software interesados en explorar todos los usos que da la gama de **SoC Zynq-7000**. Esta placa contiene un modelo de gama media-baja de la gama de SoC programables Zynq-7000, específicamente la Zedboard contiene un modelo XC7Z020-CLG484-1 del **Zynq-7000**, el modelo contiene una lógica programable equivalente a una FPGA **Artix** con 85000 celdas lógicas.

La placa contiene todo lo necesario para crear un diseño basado en varios **sistemas operativos** de corte convencional, como **Linux**, **Android** o **Windows**, sistemas operativos en **tiempo real** y la posibilidad de ejecutar código directamente sobre el hardware.

Los **componentes** y periféricos que integran la placa se pueden apreciar en la siguiente fotografía de la parte frontal de ésta, exceptuando el lector de tarjetas SD que se encuentra en su parte trasera:

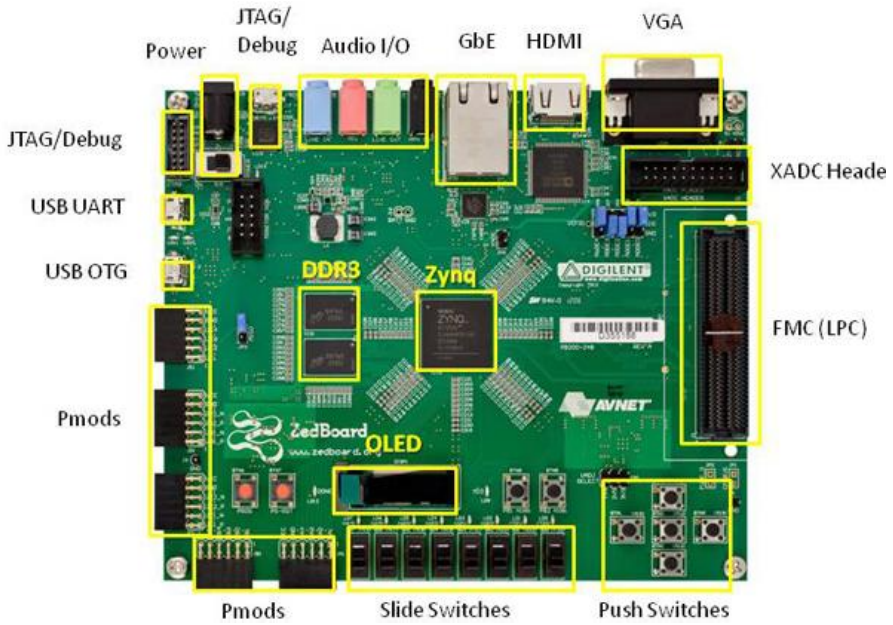


Ilustración 8: Imagen de descripción de Zedboard

Como se puede ver en la imagen anterior los componentes más importantes son los siguientes:

- SoC programable: Modelo **Xilinx** XC7Z020 **Zynq-7000** All Programmable SoC.
- Memoria:
  - **Memoria RAM**: 512 MB DDR3.
  - **Memoria Flash**: 256 Mb con interfaz QSPI.
- Interfaces:
  - **USB–JTAG**: Utilizado para la programación de la placa.
  - Interfaz de red **Ethernet 10/100/1000.**
  - **Lector de tarjeta SD**.
  - **USB 2.0 OTG**.
  - Entrada/salida serial **UART** mediante un puente USB 2.0.
  - 5 conectores para expansiones **Pmods** de Digilent.
  - Conector FMC-LPC.
  - **8 *switches*** de deslizamiento y **7 botones** de pulsación.
  - ***Jumpers*** de control.
  - **9 *leds*** para el usuario y un *led* de Done.
- Display/Audio:
  - Salida de video **HDMI**.
  - Salida de video **VGA** de 12bits de color.
  - **Pantalla OLED** de 128x32.
  - **Entrada/Salida** de **audio** y entrada de micrófono.

3.2.2 Entorno de desarrollo

El entorno de desarrollo para la implementación en la placa se instalará en un ordenador personal, específicamente de un ordenador portátil, con todo lo necesario para poder utilizar las herramientas de desarrollo de una forma eficiente y con todas las interfaces necesarias para la interconexión con la placa.

Para la elección del entorno de desarrollo software hay que especificar que la placa y el SoC dan **muchas posibilidades** entre los tipos **de entorno software** que se pueden escoger, pudiendo desarrollar en diferentes sistemas operativos, con bastante variedad de drivers y de **herramientas para el desarrollo**.

En nuestro caso el **entorno operacional software** que se ha escogido es el siguiente:

- **Sistema Operativo**: Linux Debian - Squeeze, *Kernel* 2.6.32.
- **Herramientas desarrollo**: La última versión del entorno de desarrollo de Xilinx, **ISE 14.3**.
- Drivers y herramientas USB:
  - ***Digilent cable drivers***, drivers para las conexiones de USB.
  - **Fxload**: programa para descargar firmware a dispositivos USB.
- Compiladores:
  - **Compilador** de **Gcc** de **ARM** para Linux, *ARM GNU Tools*.
  - **Compilador** y librerías **Gcc** de GNU versión 6.
- Otros:
  - **Terminal serial** *GtkTerm* para obtener entrada/salida **UART** con la placa.
  - **Parche** de para la generación de *bitstream* en XPS para ISE 14.3.

La **puesta a punto** del entorno y su configuración, se puede consultar como un anexo de la documentación del proyecto [11.1].

El **Xilinx ISE 14.3** es un entorno de desarrollo proveído por el fabricante del SoC Xilinx, este entorno de desarrollo está **orientado** en su mayoría para el **diseño de hardware y software** sobre **FPGAs** y dispositivos programables. Está disponible para varios sistemas operativos diferentes, en nuestro lugar hemos escogido un entorno de desarrollo **Linux** ya que a pesar de ser ligeramente más compleja su puesta a punto, permite una **mayor flexibilidad** y opciones.

Para el desarrollo del proyecto con la Suite Xilinx ISE 14.3 se han utilizado principalmente 3 programas de la suite de Xilinx: **ISE** y **XPS** para la creación hardware del periférico y la **Xilinx SDK** para el desarrollo del software optimizado para el SoC.

A continuación se explicarán brevemente el funcionamiento y las funcionalidades de las herramientas de desarrollo que se van a utilizar.

Xilinx Platform Studio

Xilinx **XPS** (***Xilinx Platform Studio***) permite realizar el **diseño hardware** de una forma **sencilla** e **intuitiva** mediante cómodos **asistentes** personalizados. El programa cuenta con **herramientas** para poder modificar este hardware de una forma sencilla después de haber sido creado.

También permite añadir componentes como **IPs**, **componentes hardware** con propiedad intelectual para cubrir una determinada función como ***soft processors***, controladores de interfaces o aceleradores en el caso de la gama Zynq-7000.

Para poder descargar la descripción en la placa, el programa se encarga de generar un fichero que contiene la descripción hardware con extensión "**.bit"** conocido como ***bitstream***. El programa es capaz de descargar este *bitstream* en la placa, exportarlo al XSDK, y adicionalmente cargar código compilado para la especificación hardware como un ejecutable "**.elf".**

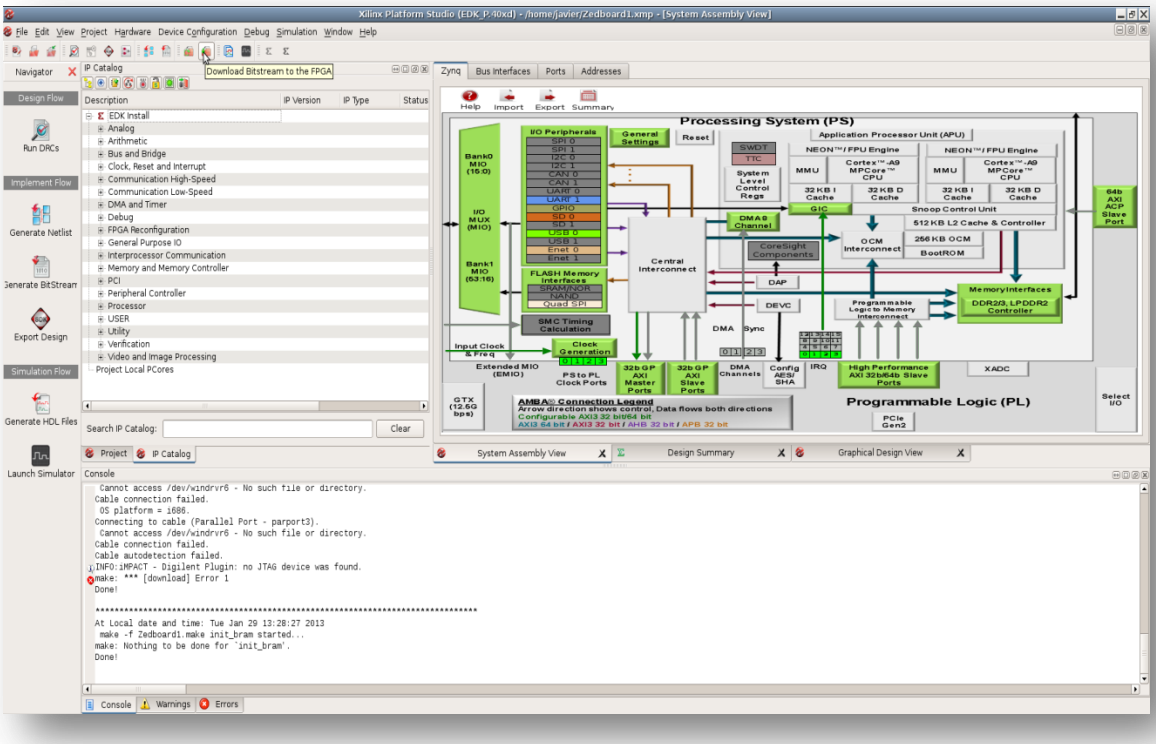


Ilustración 9: Pantalla Principal XPS

Xilinx SDK

El programa **XSDK** (***Xilinx Software Development Kit***) es la herramienta para el diseño de software (*Baremetal* o Linux).

El programa utiliza la interfaz del entorno de desarrollo **Eclipse**, y contiene casi todas las **herramientas** y **utilidades** de uno de los entornos de desarrollo más **extendidos** en la actualidad. Esto permite poder implementar código en **C/C++** como si se estuviera haciendo de una forma convencional para cargarlo directamente en un diseño hardware o para una distribución Linux.

Además de todas las herramientas habituales en un entorno de desarrollo, incluyendo la posibilidad de realizar ***debug***, el entorno permite cargar los diseños hardware mediante un fichero ***bitstream***, realizar paquetes de soporte para placas, o asignar memoria en el **mapa de memoria** mediante un fichero ***Linked Script***.

Adicionalmente, también permite descargar el diseño hardware en la lógica programable y su ejecución o *debug* desde el mismo programa.

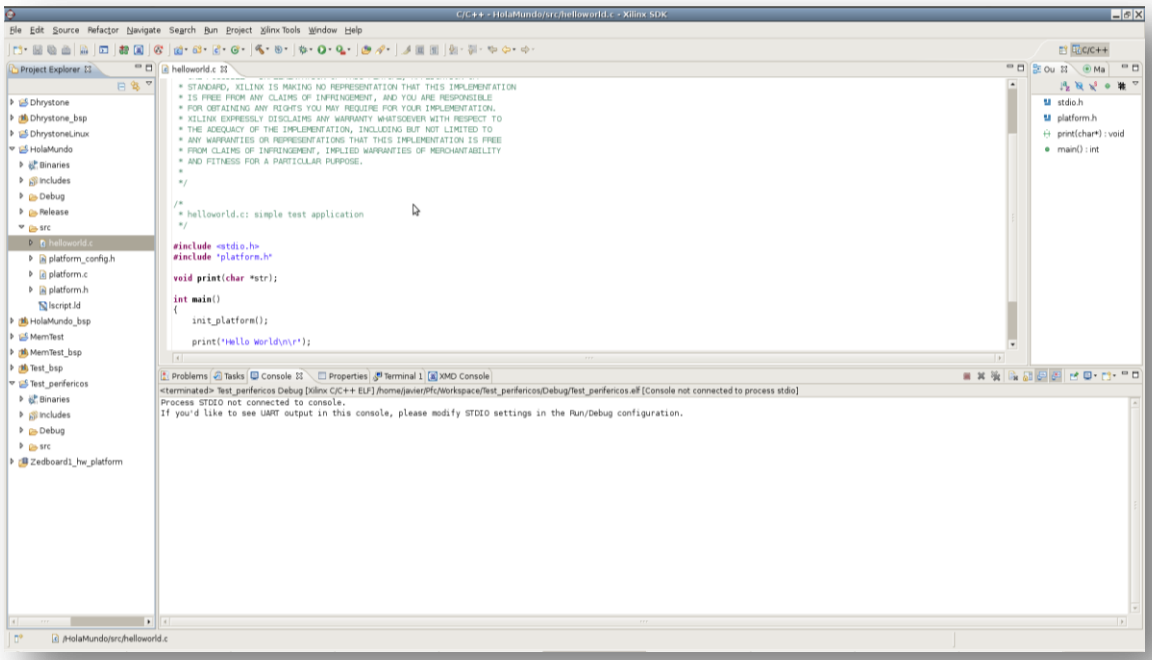


Ilustración 10: Pantalla Principal XSDK

Xilinx ISE

La herramienta de desarrollo **Xilinx ISE** (Integrated Software Environment) se trata de una herramienta utilizada para el diseño e implementación de diseño hardware. El diseño del hardware se realiza utilizando lenguajes de descripción de hardware o **HDL**, en concreto **VHDL** y **Verilog**. Adicionalmente **crear y modificar diagramas** de descripción de circuitos digitales, creados en formato **RTL**, que permiten describir el sistema con un nivel **mayor** de **abstracción** para ayudar a la compresión del código HDL que le acompaña.

La herramienta de desarrollo dispone de todos los mecanismos y fases necesarias para implementar el hardware escrito o modificado, siendo capaz de hacer la **compilación y síntesis** del diseño hardware escrito en un HDL.

Adicionalmente permite la compilación e implementación del código, también es capaz de **generar** el bitstream necesario para que sea implementado en las FPGAs y puede programarlo en la FPGA utilizando la herramienta iMPACT.

Finalmente señalar que el programa viene acompañado con **varias herramientas** para ayudar en la implementación del hardware, las más remarcables son el simulador de diseño hardware **iSIM** y el gestor y generador automático de IP cores llamado **Core Generator** El simulador iSIM permite la prueba de los diseños hardware mediante ***testbench*** y **estimulación de señales**, mostrando visualmente la salida de las señales del diseño hardware simulado. Por otro lado el generador de cores permite crear **IP** para instanciar en los diseños esquemáticos o HDL ya creados, la herramienta genera varios ficheros para facilitar la creación de un IP a partir de una especificación hardware.

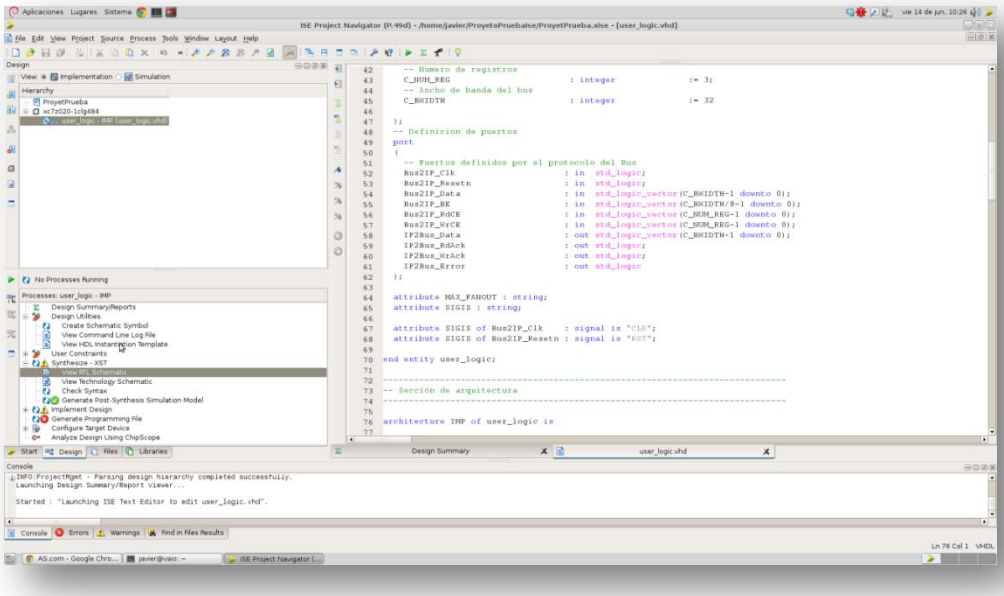


Ilustración 11: Pantalla principal ISE

3.3 Requisitos de usuario

En este apartado se **detallará** un pequeño catálogo de **requisitos de usuario** con el fin de definir, explicar y catalogar las características y restricciones que tiene que cumplir nuestro sistema tanto a nivel de software como de hardware, de una manera más explicativa que en la descripción general del sistema.

Los requisitos definirán las capacidades y restricciones que tienen de cumplir todas las partes del sistema desde el software como el hardware. Para esta descripción de los requisitos software estos se subdividen en dos tipos de requisitos, los **requisitos** de **capacidad** o **funcionales**, y los **requisitos** de **restricción** o **no funcionales**.

Los **requisitos de capacidad** definen las **capacidades** o **funcionalidades** tienen que ser capaz de realizar tanto el sistema realizado como las pruebas. Por otro lado, los **requisitos de restricción** establecen la forma en la que se tiene que realizar el sistema y las pruebas, especificando las **restriccione**s y **normas** que tienen que cumplir para su desarrollo.

Finalmente se detallarán una pequeña cantidad de **requisitos hardware**, que especifican los requisitos que tiene que cumplir el hardware para la realización del proyecto.

El formato que se va a utilizar para la especificación de requisitos es la siguiente:

Identificador	**XXX-YY**
Título	
Prioridad	☐Alta ☐Media ☐Baja
Necesidad	
Fuente	
Descripción	

Tabla 1: Plantilla de requisitos

La descripción de la plantilla de requisitos a utilizar es la siguiente

- **Identificador:** código único e identificativo de cada requisito. Consta de dos partes, XXX se refiere al tipo de requisito siendo RUC los requisitos de usuario, y RUR los requisitos de restricción. La parte YY es el número de requisito

- **Título:** es el nombre único que define al requisito.

- **Prioridad:** establece la importancia del requisito, en función de la necesidad a la hora de realizar el proyecto. Los valores admitidos son alta, media o baja

- **Necesidad:** es la importancia del requisito desde el punto de vista del cliente. Los valores posibles son **esencial**, **conveniente** u **opcional**

- **Fuente:** persona o grupo de personas donde procede el requisito.

- **Descripción:** descripción detallada y completa del significado del requisito.

3.3.1 Requisitos de capacidad

Identificador	RUC-01
Título	Tipo de periférico
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
El periférico a implementar tiene que ser capaz de realizar sumas de dos números enteros binarios de 16 Bits.	

Tabla 2: RUC-01

Identificador	RUC-02
Título	Interconexión periférico y sistema
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
La interconexión entre el periférico creado y el resto del sistema será mediante un bus del sistema. Debido a esto las interfaces del periférico tienen que ser las entradas y salidas de la especificación del bus.	

Tabla 3: RUC-02

Identificador	RUC-03
Título	Separación lógica y comunicación periférico
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Conveniente
Fuente	Cliente
Descripción	
Es recomendable que el periférico contenga una capa intermedia para traducir la entrada/salida recibida a la lógica de usuario, para facilitar la implementación de la lógica del sumador. De este modo separar la implementación de la lógica del sumador con la comunicación con el resto del sistema.	

Tabla 4: RUC-03

Identificador	RUC-02
Título	Periférico como IP Core/ Block
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
El periférico tiene que ser reutilizable y con propiedad intelectual, tiene que ser lo que se conoce como IP Core/Block. El dispositivo tiene que ser válido para introducirse como un bloque para FPGAs.	

Tabla 5: RUC-02

Identificador	RUC-03
Título	Validez del diseño hardware del System on Chip
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
El diseño hardware completo del System of Chip tiene que ser válido para la implementación en un SoC programable Zynq-7000.	

Tabla 6: RUC-03

Identificador	RUC-04
Título	Importación periférico en sistema
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
El diseño hardware del SoC tiene que ser capaz de importar bloques con propiedad intelectual o IP blocks. Esto es necesario para permitir importar en el sistema el periférico creado.	

Tabla 7: RUC-04

Identificador	RUC-05
Título	Acceso software a información del sumador
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
El sistema tiene que permitir al desarrollador software poder interactuar con el periférico. El sistema tiene que controlar el acceso y permitir operaciones de escritura lectura sobre los datos del periférico.	

Tabla 8: RUC-05

Identificador	RUC-06
Título	Formato de pruebas de simulación
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
Se realizarán pruebas de simulación sobre el periférico diseñado. Estas pruebas comprobarán su correcto funcionamiento analizando el comportamiento del periférico ante un estímulo externo, utilizando estimulación de señales.	

Tabla 9: RUC-06

Identificador	RUC-07
Título	Formato de pruebas de simulación
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
Las pruebas de simulación tienen que comprobar el correcto funcionamiento de su entrada/salida, de sus controles de acceso y del proceso de suma.	

Tabla 10: RUC-07

Identificador	RUC-08
Título	Definición pruebas sobre el periférico
Prioridad	☐Alta ☑Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
Las pruebas software a realizar sobre el periférico se lanzarán directamente sobre el hardware del sistema si la necesidad de un SSOO. Estas pruebas comprobarán la correcta capacidad de acceso al periférico desde una implementación software.	

Tabla 11: RUC-08

Identificador	RUC-09
Título	Acceso a registros periférico
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
El sistema tiene que permitir de acceder al periférico y la información de sus registros a la hora de realizar el desarrollo software.	

Tabla 12: RUC-09

Identificador	RUC-10
Título	BSP para compilación
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
Es necesaria la creación de una plataforma de soporte o BSP para la compilación del código directamente sobre el hardware.	

Tabla 13: RUC-10

3.3.2 Requisitos de restricción

Identificador	RUR-01
Título	Plataforma de Trabajo
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
Todo el diseño, implementación y prueba se tiene que realizar utilizando las herramientas provistas por la suite de desarrollo Xilinx ISE 14.3	

Tabla 14: RUR-01

Identificador	RUR-02
Título	Definición herramienta diseño HW periférico
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
La completa implementación del código hardware del periférico se realizará íntegramente con la herramienta ISE.	

Tabla 15: RUR-02

Identificador	RUR-03
Título	Formato diseño hardware periférico
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
El diseño hardware del periférico se realizará utilizando únicamente los lenguajes de descripción de hardware Verilog o VHDL.	

Tabla 16: RUR-03

Identificador	RUR-04
Título	Creación plantilla IP
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
El diseño hardware se realizará partiendo de una plantilla creada por el entorno de desarrollo para la creación de IP Cores especialmente diseñados para el Zynq.	

Tabla 17: RUR-04

Identificador	RUR-05
Título	Formato pruebas simulación
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
Las pruebas de simulación se diseñarán como ficheros testbench escritos en el lenguaje de descripción de hardware VHDL. Estos testbench una vez compilados se ejecutarán en el simulador de ISE, iSim.	

Tabla 18: RUR-05

Identificador	RUR-06
Título	Protocolo de interconexión Periférico
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
La interconexión entre el sistema y el periférico se realizará mediante un bus de datos que utiliza la especificación definida por el protocolo AXI4Lite.	

Tabla 19: RUR-06

Identificador	RUR-07
Título	Diseño Sistema completo hardware
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
La creación del hardware del sistema completo utilizando la herramienta de desarrollo ISE. Asimismo también se configurará la interconexión entre el periférico y el sistema, y la asignación de puertos y direcciones de memoria con la misma herramienta.	

Tabla 20: RUR-07

Identificador	RUR-08
Título	Exportación de descripción del hardware del Sistema
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
Para la exportación de la descripción hardware de la totalidad del sistema, se generará un fichero conocido como *bitstream* que describe el sistema. Este fichero se utilizará para que posteriormente se pueda descargar e implantar en la placa.	

Tabla 21: RUR-08

Identificador	RUR-09
Título	Implantación del diseño hardware
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
La implantación del diseño hardware se realizará descargando y programando la placa con el fichero *bitstream* generado. La descarga del bitstream en la placa se realizará mediante el puerto JTAG de la placa.	

Tabla 22: RUR-09

Identificador	RUR-10
Título	Implantación del diseño hardware
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
La implantación del diseño hardware se realizará descargando y programando la placa con el fichero *bitstream* generado. La descarga del bitstream en la placa se realizará mediante el puerto JTAG de la placa.	

Tabla 23: RUR-09

Identificador	RUR-10
Título	BSP diseño software
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
El BSP necesario para la compilación de software ejecutable *baremetal* se generará automáticamente a partir del fichero de descripción del sistema o *bitstream*.	

Tabla 24: RUR-10

Identificador	RUR-11
Título	Lenguaje Programación diseño software
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
Para todo el diseño software realizado se utilizará el lenguaje de programación C/C++. Posteriormente se compilará el código adaptado al hardware creado utilizando una herramienta de compilación cruzada y utilizando un BSP.	

Tabla 25: RUR-11

Identificador	RUR-12
Título	Entrada y salida estándar
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
La entrada y salida estándar de la implementación software será mediante un puerto serie.	

Tabla 26: RUR-12

3.3.3 Requisitos Hardware

Identificador	RHW-01
Título	Zynq en placa de desarrollo
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
Es necesario que la placa de desarrollo donde se realicen las pruebas contenga un SoC programable Xilinx Zynq-7000.	

Tabla 27: RHW-01

Identificador	RHW-02
Título	Puertos placa de desarrollo
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
Es necesario que la placa de desarrollo contenga de los puertos JTAG, para la programación de la placa, y UART2USB.	

Tabla 28: RHW-02

Identificador	RHW-03
Título	Puertos USB equipo de desarrollo
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
Es necesario que el equipo de desarrollo contenga al menos un puerto USB para poder utilizar la conexión JTAG y UART para la comunicación con la placa.	

Tabla 29: RHW-03

Identificador	RHW-04
Título	Características técnicas entorno desarrollo
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Esencial
Fuente	Cliente
Descripción	
Es conveniente que el equipo de desarrollo tenga una memoria RAM de 2GB o superior, y un espacio de almacenamiento de 15GB para la instalación y el funcionamiento del entorno de desarrollo.	

Tabla 30: RHW-04

Identificador	RHW-05
Título	Potencia equipo de desarrollo
Prioridad	☑Alta ☐Media ☐Baja
Necesidad	Conveniente
Fuente	Cliente
Descripción	
Es recomendable que el equipo de desarrollo contenga un procesador de gran potencia para reducir el tiempo en la compilación e implementación de los diseños HW y SW. En particular, es recomendable para la generación del *bitstream* que requiere una gran capacidad de procesamiento.	

Tabla 31: RHW-05

4 Diseño

En este apartado se explicará el proceso de diseño realizado para todas fases del proyecto, para ello se explicará el **funcionamiento** del **diseño HW y SW** del sistema y del periférico y sus pruebas asociadas, especificando las **consideraciones de diseño**, su **funcionamiento** y **composición**.

4.1 Diseño Periférico

El periférico que se ha diseñado se trata de un **sumador** de 32 bits. Principalmente este sumador se encarga de sumar dos números almacenados en dos registros y guardar la suma obtenida en otro registro. El sumador se comunica con el exterior, en nuestro caso con el Zynq, como un dispositivo **esclavo**, mediante un bus que se ciñe al protocolo **AXI4LITE**.

Para la creación del diseño hardware del periférico se ha basado en la **plantilla** que provee la suite de desarrollo para la **creación de periféricos** como **IP cores**. Esta plantilla provee una arquitectura básica para facilitar la gestión de memoria con el sistema, y contiene librerías diseñadas para acceder a los registros del periférico mediante direcciones de memoria.

A continuación se muestra un diagrama con el funcionamiento general del periférico:

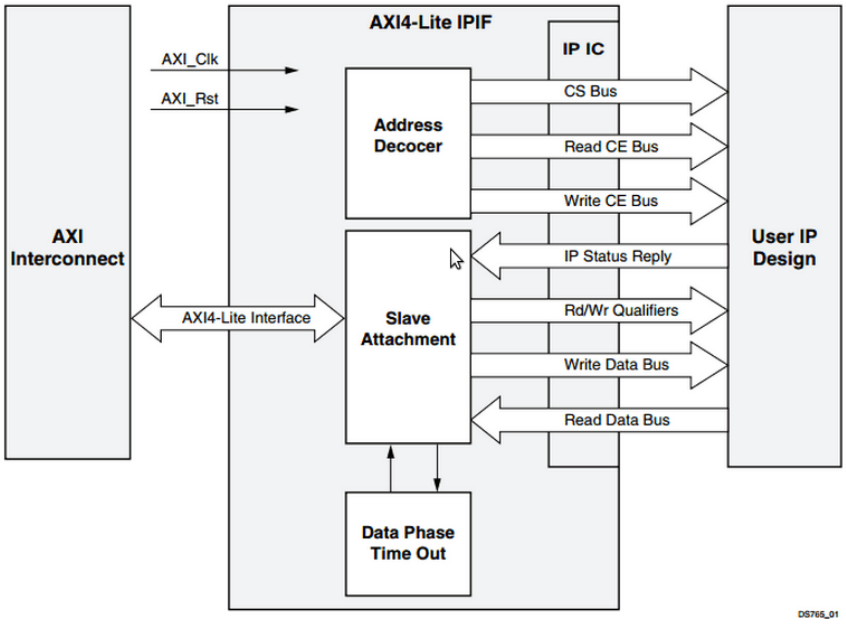


Ilustración 12: Diagrama de diseño de periférico

Como se puede observar en el diagrama, el periférico está compuesto principalmente por dos módulos, el módulo axi_lite_IPIF y la lógica de usuario o diseño del IP.

El módulo **axi_lite_IPIF**, se trata de un módulo **autogenerado** por las herramientas de la suite que se encarga de **gestionar** el acceso al periférico mediante **direcciones de memoria**, y se encarga de traducir los datos recibidos por el bus a la lógica de usuario.

Por otro lado, la **lógica de usuario** será la encargada de realizar el proceso de **suma**, almacenando la suma de dos registros en otro registro, así como realizando el control de acceso a estos registros.

A continuación se explicará más detalladamente el funcionamiento de los dos módulos.

4.1.1 Axi_lite_IPIF

El módulo axi_lite_IPIF se trata de un módulo provisto y generado por la herramienta de la suite **logiCORE**, y es utilizado para facilitar la implementación de una lógica de usuario de una manera sencilla. Este módulo se puede autogenerar mediante las herramientas de la suite ISE, pudiendo realizar una implementación propia para el uso de las librerías y el funcionamiento del módulo.

La implementación del componente axi_lite_IPIF se trata del **componente de más alto nivel** del periférico que se ha implementado. Este módulo principalmente se dedica a **instanciar las librerías** y la **lógica de usuario**. Además se encarga de **mapear la información** que recibe y enviarla a la lógica del usuario utilizando las librerías provistas para su traducción. Esto se realiza para **facilitar** la **gestión de información** para la lógica de usuario, tratando con registros en vez de direcciones de memoria.

El módulo está compuesto por **tres componentes** diferentes para realizar este trabajo de intermediario entre el resto del sistema y la lógica de un periférico:

- **Slave Attachment:** Este componente del módulo es el encargado de controlar el **acceso** de **lectura** o **escritura** a los **datos**, se encarga de gestionar las señales de control definidas por el protocolo y de instanciar el decodificador y el controlador de tiempo máximo de espera. También es el encargado de mapear todas las señales entre el bus, el decodificador de direcciones y la lógica de usuario.
- **Address Decoder:** Este componente es el encargado de **seleccionar** los registros con los que se quiere hacer una operación de lectura/escritura. Para ello el componente asigna direcciones de memorias a un número de registros que necesite la lógica, ambos valores son definidos mediante atributos del periférico.

Siempre que recoja algún acceso a esas direcciones asignadas a los registros, se encargará de comunicar a la lógica mediante un **selector de chips** o **registros**, qué registro o qué dirección de memoria de la lógica se quiere realizar una operación de entrada/salida.

- **Data Phase Timer:** Este componente se encarga de **comprobar** el **tiempo** que ha pasado desde que se ha realizado una petición de entrada/salida a la lógica de usuario, **finalizando** la operación si se traspasa el tiempo definido por un **atributo** del periférico.

El funcionamiento del módulo consiste en **convertir** el acceso a una **dirección memoria** a una serie de **registros** o a un grupo de **direcciones** de **memoria locales.** Esto permite abstraerse de los protocolos de comunicación y de la gestión de comunicación con el resto del sistema mediante un bus. Para ello provee a la lógica de usuario una serie de interfaces más sencillas que el protocolo de comunicación con el resto del sistema.

La implementación del módulo al tratarse **del componente de mayor nivel** contiene todas las interfaces correspondientes al protocolo axi_lite, ya que es el encargado de intercomunicarse con el resto del sistema. En nuestra implementación del módulo, solamente se contendrán las interfaces de esclavo del protocolo axi4lite, al haber sido configurado de esta manera. Además debido a esto, este componente es el encargado de contener los parámetros del periférico, definidos mediante una serie de constantes o **señales genéricas**, que permiten configurar el periférico que se ha realizado.

Las interfaces de la implementación del módulo IPIF nombrada como “mysum”, se pueden observar en el siguiente diagrama RTL.

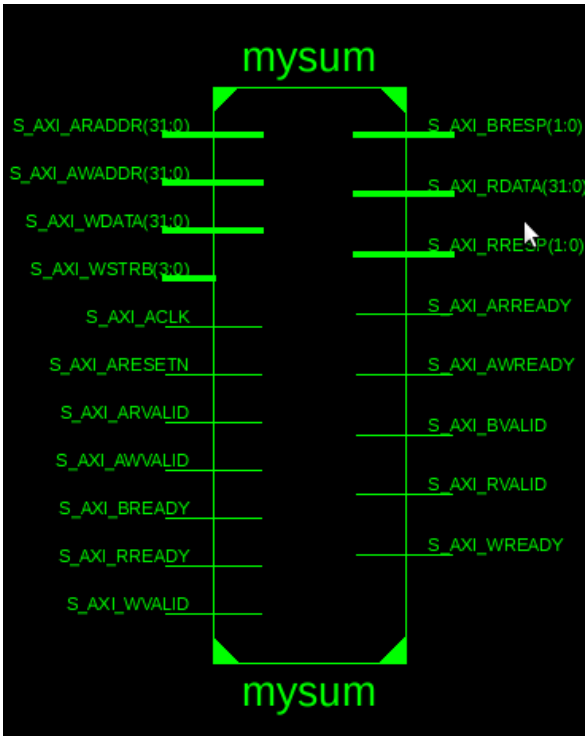


Ilustración 13: Diagrama RTL de la implementación del módulo IPIF

Para conocer más **información** sobre los **atributos** del módulo e información detallada sobre las **interfaces** del bus y las provistas a la lógica del usuario, se puede consultar en el anexo 0.

4.1.2 Lógica de usuario

La lógica de usuario se trata de un fichero de VHDL que contiene la lógica de usuario responsable de gestionar el acceso a sus registros, y realizar el **proceso de suma**. Este proceso se facilita gracias a la utilización de la librería "axi4lite ipif" explicada en el punto anterior.

Este sumador recibe la información por medio de las interfaces de entrada y salida provistas por la librería anterior. A continuación se muestra un diagrama RTL en el que se puede apreciar el diseño de la lógica de usuario con sus interfaces:

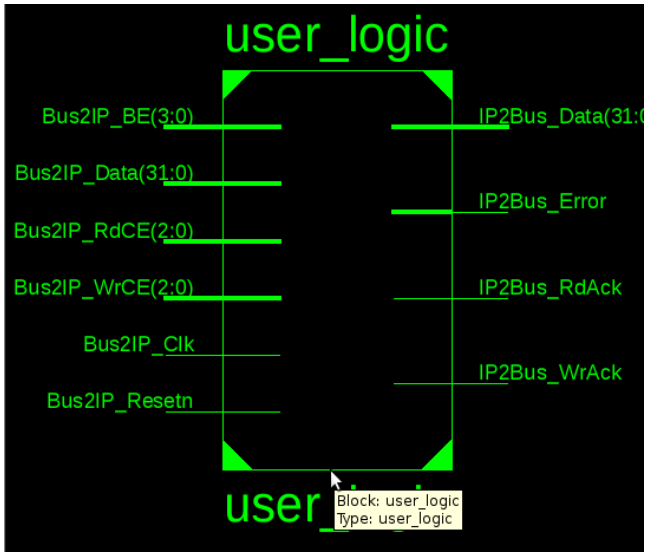


Ilustración 14: Diagrama RTL de la lógica de usuario

La entidad está compuesta por **dos procesos**, uno se encarga de la gestión de **suma y escritura** de registros, y otro de la gestión de **lectura**. Los **selectores del registro** se reciben a través de los puertos **"Bus2IP_RdCE"** y "**Bus2IP_WrCE"** mediante un código de bits**.** El sumador controla que sólo se pueda escribir o leer un registro al mismo tiempo.

Sumando 1	Sumando 2	Resultado

Ilustración 15: Selector Registro Bus2IP RdCE - WrCE

Por lo tanto las diferentes combinaciones válidas para la escritura o lectura de registros es la siguiente:

- **‘100’**: Permite escribir o leer en el primer sumando.
- **‘010’**: Permite escribir o leer en el segundo sumando.
- **‘001’**: Permite leer el resultado. En este caso la escritura no es útil permitirla debido a que siempre va a corresponder a la suma de sus sumandos.
- **‘000’**: No se quiere leer ni escribir en ningún registro, debido a eso se realizará en este momento la suma de registros.

A continuación, se explicarán los algoritmos de los procesos de lectura y escritura.

Proceso de escritura

El proceso de escritura se encargará de **gestionar la escritura** en los sumandos datos que se reciban por las interfaces. También es el encargado de realizar la suma cuando no se vaya a realizar ninguna escritura. El proceso se ejecutará siempre que llegue un **flanco alto** de la señal de **reloj** “Bus2IP_Clk”.

El algoritmo que sigue el proceso es el siguiente:

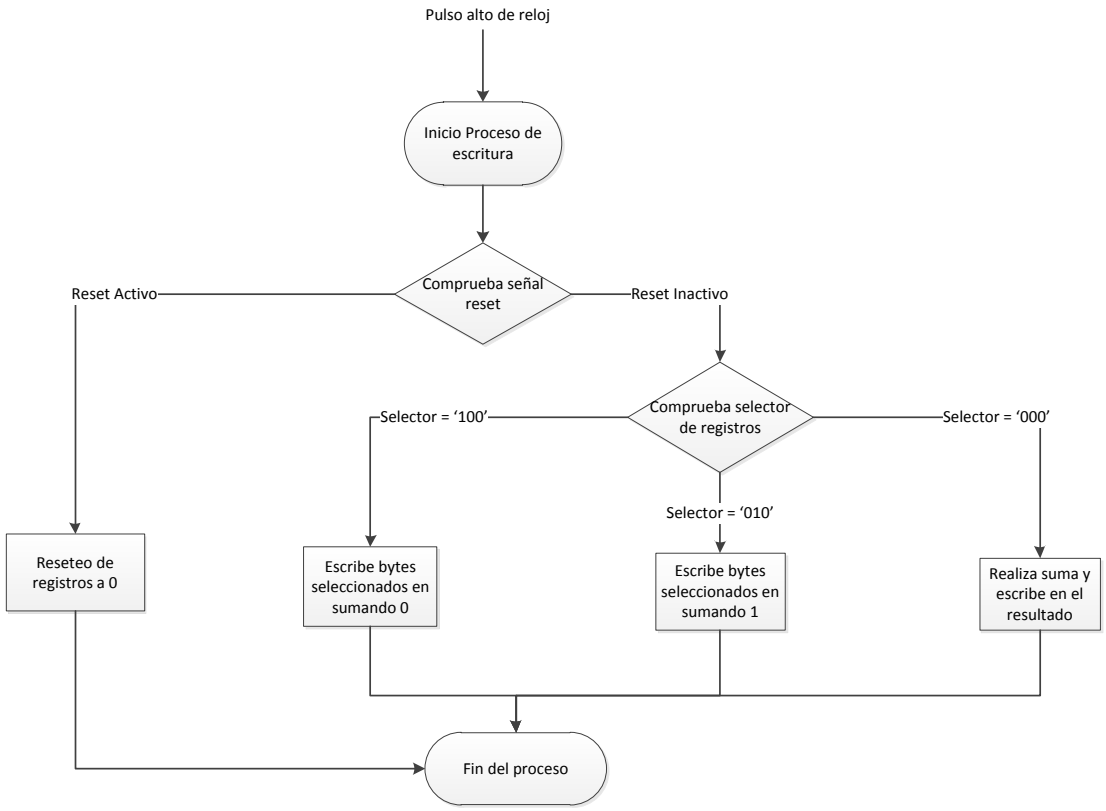


Ilustración 16: Diagrama de algoritmo de escritura en lógica de usuario

Proceso de lectura

El proceso de lectura se encarga de permitir leer de cualquiera de los tres registros que contiene la lógica de usuario. El proceso se ejecutará siempre que se haya **modificado** uno de los **registros** o el **selector de registros**, este proceso se ejecutará en dos situaciones:

- **Ejecución por cambio de selector:** Es el caso más general de ejecución del proceso, se ejecuta cuando se selecciona un registro al que leer y se muestra el valor actual.
- **Ejecución por cambio registros:** Este caso se ejecuta cuando se modifica el registro durante el tiempo de proceso de lectura desde su componente más alto, el gestor de esclavos, y el selector sigue activo. Como no ha terminado la lectura, actualiza el valor del registro reenviándolo por el puerto.

El algoritmo seguido por el proceso es el siguiente:

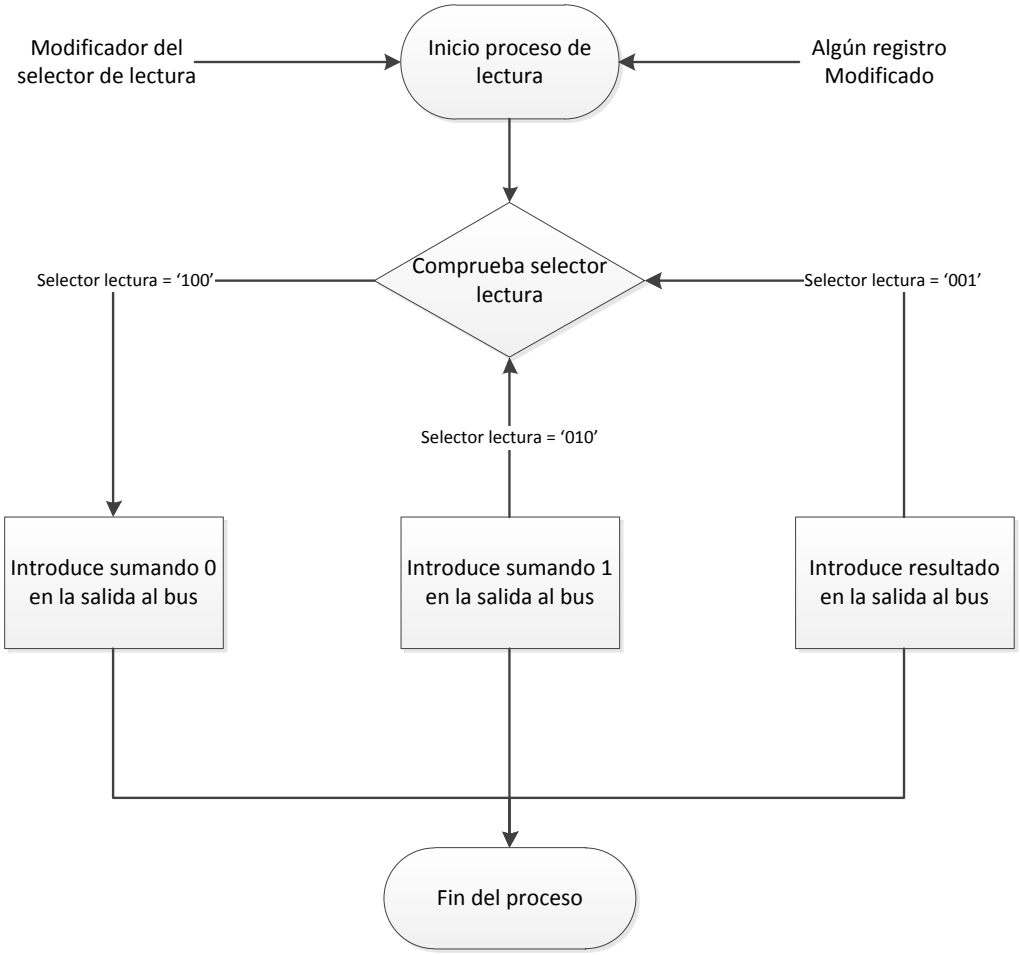


Ilustración 17: Diagrama de algoritmo de lectura en lógica de usuario

4.2 Diseño de pruebas unitarias

Para comprobar el correcto funcionamiento del módulo hardware diseñado, se han realizado varias **pruebas de simulación**. Estas pruebas han sido implementadas como *testbench* en lenguaje VHDL, utilizan la técnica de estimulación de señales, y son ejecutadas utilizando el simulador iSIM.

Estas pruebas se han diseñado para comprobar el correcto funcionamiento de los principales componentes del módulo hardware desarrollado, antes de introducirlo dentro del sistema. Para ello se han diseñado diversas pruebas para comprobar su correcta lectura y escritura, y para comprobar el buen funcionamiento de la lógica de usuario.

Dichas pruebas se basan en **simular señales de entrada** de estos componentes, enviándolas a través de sus interfaces, y comprobar que las señales de salida obtenidas son las deseadas.

Para comprobar el funcionamiento del sumador, se realizarán **dos tipos** de pruebas unitarias en formato de *testbench*. En primer lugar se han realizado pruebas para comprobar el correcto funcionamiento de la **lógica de usuario**, mediante la estimulación a sus interfaces reducidas, para verificar el buen funcionamiento de la lógica y el valor de la suma. El otro tipo de pruebas se harán sobre el **módulo completo**, estimulando y monitorizando el bus del sistema al que se va a conectar, este tipo de pruebas servirán para comprobar que se ha configurado correctamente el componente "axi4-lite ipif", y si el periférico funciona correctamente para ser importado en el sistema.

Para todas las pruebas unitarias se ha simulado una **señal de reloj asíncrona** que cambia cada **100 ns**, para ello se ha realizado un proceso de reloj en todos los *testbench*, menos en el último que se aumentará la frecuencia del reloj. Esta señal de reloj utilizado corresponde a una señal de una frecuencia de **10 MHz.**

Las pruebas unitarias realizadas sobre la lógica de usuario son las siguientes:

- **Testbench 1**: En esta prueba básica se comprobará el correcto funcionamiento de la **entrada/salida** sobre un registro. Para ello se le enviará un dato por sus interfaces y se comprobará que la salida al leer el registro sea el mismo dato.
- **Testbench 2**: En esta prueba se comprobará de una forma más avanzada el funcionamiento de la lógica de usuario y la gestión de escritura-lectura de registros. Se revisará que funciona el **selector de bytes** de escritura en una palabra. Para ello se escribirá un dato en el registro 2, y posteriormente se escribirá varios bytes de otro dato en el mismo registro, y se comprobará que sólo se ha escrito en los bytes elegidos.
- **Testbench 3**: En esta prueba se comprobará principalmente el perfecto funcionamiento del sumador en general. Para ello se **escribirán** datos en los **sumandos** que equivalen a los dos primeros registros y se comprobará que se realiza la suma correctamente, realizando la **lectura** del resultado del tercer registro.

Una vez comprobado que la lógica de usuario está correctamente implementada, se han realizado unas **pruebas sobre el periférico completo**. Al ser estas pruebas algo más complejas, se explicarán más detalladamente a continuación.

Testbench 4

En esta prueba se verificará la escritura correcta en un registro simulando una **escritura** de un dato a una **dirección de memoria** mediante el **bus** del sistema. Para ello se elegirá un dato que se escribirá en la **primera dirección de memoria asignada** a los registros, es decir, se intentará escribir en el primer registro.

Para realizar la escritura se ha necesitado estimular en orden todas las señales requeridas para realizar una escritura de un dato en una dirección de memoria, las señales de *ready* para la dirección de memoria y los datos entre otras cosas.

Testbench 5

Esta prueba se realizará para comprobar una correcta **lectura** de un registro, accediendo a la dirección de memoria asignada a él, para ello en primer lugar se simulando una **escritura** de un dato a otra **dirección de memoria** mediante el **bus** del sistema. Se elegirá un dato que se escribirá en la **segunda dirección de memoria asignada** a los registros, es decir, se intentará escribir en el segundo registro, y posteriormente se realizará una lectura sobre esa dirección de memoria. Adicionalmente se realizará una escritura en la que se **desactivará un byte** de la palabra para comprobar su correcto funcionamiento.

Basándonos en la prueba anterior, se simulará una **petición de escritura** en una dirección de memoria asignada al periférico, en particular al segundo registro, con la particularidad de que no se escribirá el **primer byte** del dato recibido por el bus.

Posteriormente cuando llegue la señal indicando que se ha realizado correctamente la **escritura del dato**, se introducirá la **dirección de memoria** a la que se quiere acceder y en orden las señales de control necesarias, simulando un acceso de lectura estándar al periférico.

Para finalizar se realizará una **escritura** en ese registro únicamente sobre el **byte más significativo** del dato introducido, siguiendo el mismo procedimiento que anteriormente. Finalmente se comprobará mediante una **lectura** que dato corresponde al dato inicial introducido de entrada excluyendo al primer byte que tendrá el último valor introducido.

Testbench 6

Este ***testbench*** se encarga de comprobar que la suma se realiza correctamente en la lógica de usuario. Para ello se escribirán dos datos en los dos registros **sumandos** de entrada con la configuración necesaria de **escritura** comentadas anteriormente.

Posteriormente cuando llegue la señal indicando que se han realizado las escrituras correctamente, se realizará una **lectura** sobre la dirección de memoria del valor del **resultado**, almacenado en el tercer registro, comprobando que la suma se realizado de una manera correcta.

Testbench 7

En este testbench se probará hacer una mezcla de todas las pruebas anteriores, probando lecturas, escrituras, reseteos, selección de bytes. Adicionalmente se subirá el periodo del reloj creado, **aumentando la frecuencia de reloj**. Esto se ha realizado como una prueba de *stress* igualando la frecuencia del reloj simulado a la frecuencia usual del reloj en la placa Zedboard.

La frecuencia que normalmente utiliza el reloj implantado en la placa Zedboard corresponde a una frecuencia de 100 MHz. Por lo tanto se reducirá el periodo con el que se ejecuta el reloj, el reloj pasará de periodo de 100 ns por ciclo a un periodo de 10 ns. Para calcular el periodo se ha utilizado la fórmula de relación entre periodo y frecuencia

$$T = \frac{1}{f}.$$

4.3 Diseño Arquitectura Sistema

La arquitectura del procesador escogida se basa en un **diseño sencillo** del **SoC** Zynq. Para ello se ha implementado un **sistema sencillo** para el SoC, en el que la casi única decisión de diseño radica en cómo y dónde introducir el periférico creado e importado en el diseño del SoC.

El sistema del procesador que se ha realizado se trata de un **diseño** básico como **plantilla** para el modelo del SoC Zynq7000 de la Zedboard. Con este diseño se han definido las **frecuencias** por defecto de los procesadores y **APUs**, así como los buses, interfaces y direcciones de memoria para la memoria principal y el sistema de procesamiento en general.

No se ha realizado ningún cambio en el multiplexador de entrada/salida, el dispositivo **MIO.** Esto es debido a que inicialmente no se quiere gestionar la entrada/salida del periférico desde el MIO, y no es necesario realizar ningún cambio en el multiplexado para el resto del sistema, ya que no se quiere realizar **ninguna gestió**n de **interrupciones**.

El diseño del sistema realizado se podría resumir en las dos partes por las que está compuesto el Zynq, el sistema de procesamiento o **PS** y la lógica programable o **PL**.

Se ha conectado el periférico a un bus de datos del protocolo AXI4Lite funcionando como un esclavo. Adicionalmente recibe la señal del reloj recibida directamente desde el bus de la sección PS.

El acceso a los registros que contiene el periférico se realiza mediante el uso de direcciones de memoria, en un rango definido por sus constantes "BASE_ADDRESS" y "HIGH_ADDRESS". Se le asignará un rango de memoria entre esas dos constantes, como un **mapa de memoria compartido** con el resto del sistema, y que corresponderán al interior del periférico, en nuestro caso los registros de este periférico.

Las direcciones que se han asignado al periférico son las direcciones inmediatamente superiores a la memoria principal, es decir, el rango siguiente de direcciones después de la memoria principal. Como se ha indicado en el diseño del periférico según se indica en una constante, el periférico requiere al menos por defecto un **direccionamiento mínimo** de **4Kb**, debido a que se tienen un gran número de direcciones de memoria disponibles, con este tamaño de palabra se tienen $\mathbf{2^{32}} = \mathbf{4.294.967.296}\ \boldsymbol{direcciones\ de\ memoria,}$ y debido a que solo se tiene 1GB de memoria principal utilizada, se ha decidido no cambiar el valor de esta constante.

El rango de memoria escogido para el periférico es el siguiente:

$$0x7E400000 - 0x7E400FFF$$

5 Implementación

En esta sección del documento se explicarán todos los **pasos realizados** para la implementación del sistema. Para explicar todo el desarrollo realizado, se utilizarán **capturas de pantalla** para ayudar a la comprensión del desarrollo realizado.

Para el desarrollo del sistema y las pruebas se han realizado **varias fases** distintas, así como diferentes elementos y componentes. Estas fases han sido ordenadas cronológicamente, y en general necesitan de la fase anterior para su desarrollo.

Las fases por orden cronológico son las siguientes:

1. **Puesta a punto del sistema:** En esta fase se realizará la instalación y configuración del entorno de desarrollo Para conocer la puesta a punto del sistema se puede revisar en el **Anexo 1** del documento [11.1].
2. **Creación diseño Zynq7000:** En esta fase se explicará el proceso realizado para la **implementación del diseño** general del **SoC**. Para la creación del sistema se utilizará la herramienta **XPS**, permitiendo implementar el Zynq7000 siguiendo sencillos asistentes.
3. **Creación del periférico:** En esta fase se explicará cómo se ha formado la plantilla para la **creación del IP** sumador que se ha diseñado, utilizando la herramienta **XPS**. Mediante varios asistentes se configurarán los valores para la creación de un proyecto de la herramienta ISE, que servirá para la implementación del periférico como una plantilla. Además de la creación de la plantilla se configurará ligeramente el componente **axi4lite_ipif** que se generará para ayudar a implementar la lógica de usuario.
4. **Implementación del periférico:** En esta fase se detallará el proceso seguido para la **implementación del diseño HW del periférico** partiendo de la plantilla generada anteriormente. Una vez completada la implementación del diseño se realizarán los pasos necesarios para ser importado al sistema completo, estos pasos tratan principalmente de la **compilación, la síntesis** y finalmente ***el place and route*** del diseño HW del periférico.
5. **Importación del periférico:** En esta fase se explicará el proceso realizado para **importar** el **periférico diseñado** y correctamente **sintetizado**, y agregarlo a la totalidad del sistema creado. En primer lugar nos encargaremos de configurar los valores iniciales y constantes que definen el periférico. Y posteriormente nos encargaremos de **configurar** la **interconexión** con el resto del sistema así como **gestionar** su rango de direcciones de memoria asignado.
6. **Exportación del sistema:** En esta breve fase se comentará el proceso realizado para poder realizar la exportación del sistema completo como un *bitstream*. Para posteriormente usarse en la **programación de la placa**, o poder realizar el **BSP** necesario para compilar cualquier tipo de **diseño software** a partir de esta exportación.
7. **Creación SW:** En esta fase se explicarán todos los pasos seguidos para poder implementar, compilar y posteriormente ejecutar un diseño software, que se pueda ejecutar directamente sobre el diseño hardware implementado.

Para ello partiendo de la exportación HW, se creará una plataforma HW necesaria para que se pueda compilar el código diseñado, que posteriormente pueda ser ejecutada por la placa.

8. **Implantación HW y SW:** En esta fase se explicarán detalladamente todo el proceso necesario para poder **descargar un diseño HW** en la placa. Se explicará cómo se ha realizado la programación de la placa, y se explicarán brevemente el funcionamiento de las diferentes alternativas de la programación.

5.1 Implementación hardware sistema Zynq-7000

Para la implementación del diseño HW del SoC se va a realizar utilizando la herramienta **XPS** principalmente. En primera instancia a la hora de arrancar el programa XPS se muestra un menú donde se permite gestionar los proyectos, o ver documentación del programa. En nuestro caso se comenzará creando un nuevo proyecto usando el asistente.

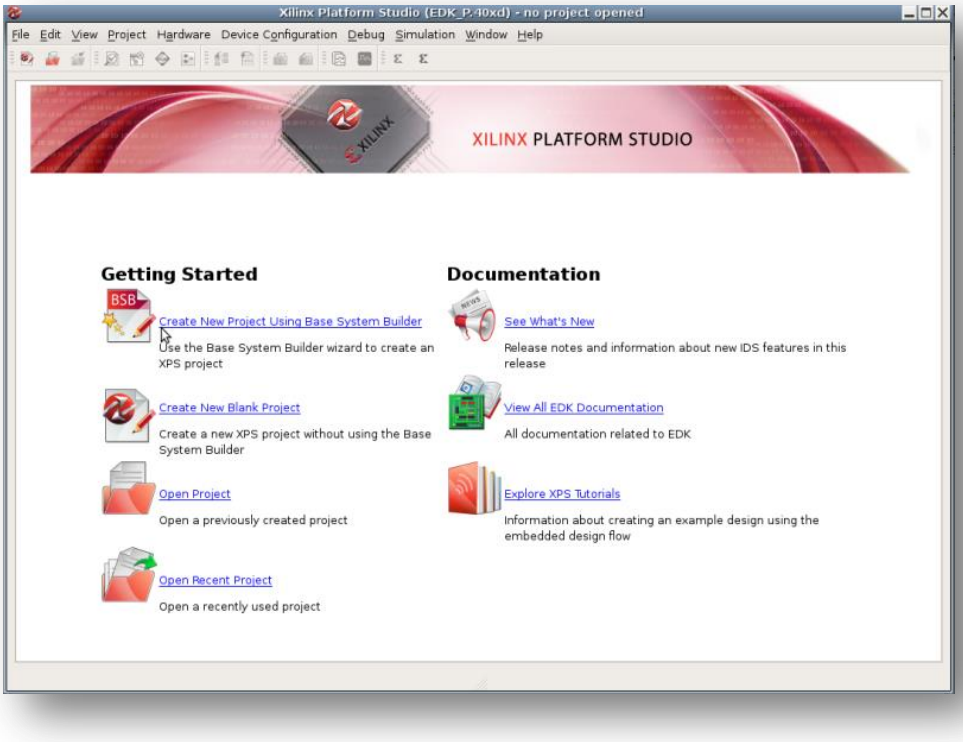


Ilustración 18: Pantalla Inicial XPS

El asistente lo primero que pide es introducir el nombre y la ruta del proyecto, y posteriormente elegir el tipo de bus de interconexión dentro del diseño hardware. Como se ha explicado anteriormente, se utilizará una **interfaz AXI** perteneciente al protocolo de **interconexión AMBA**.

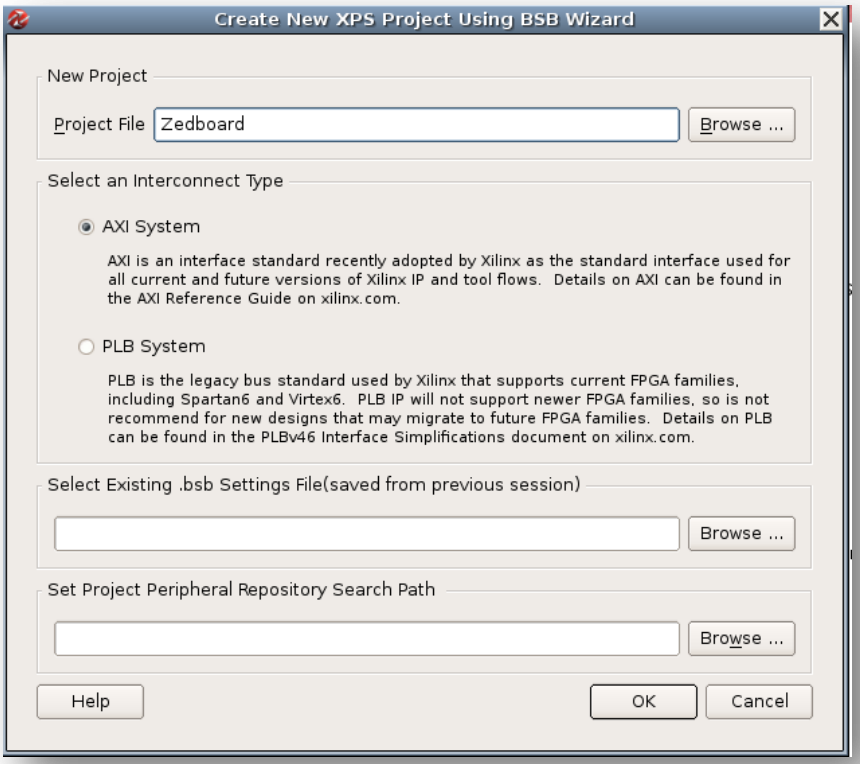


Ilustración 19: Creación de un nuevo proyecto en XPS

Adicionalmente se puede elegir un **BSP** de inicio sobre el que basar el proyecto. También se ofrece la posibilidad de poner una ruta en el que buscar **drivers extra**, que no se encuentren en la raíz de la suite, para facilitar la implantación del diseño en la placa.

Posteriormente se elegirá el tipo de **hardware** que se va a diseñar. Aunque el sistema da la posibilidad de hacerlo de una manera personalizada, se ha optado por utilizar la **plantilla** hecha para nuestro SoC. Por lo tanto, se elegirá la **Zedboard** de **Avnet**, que contiene el SoC programable Zynq-7000.

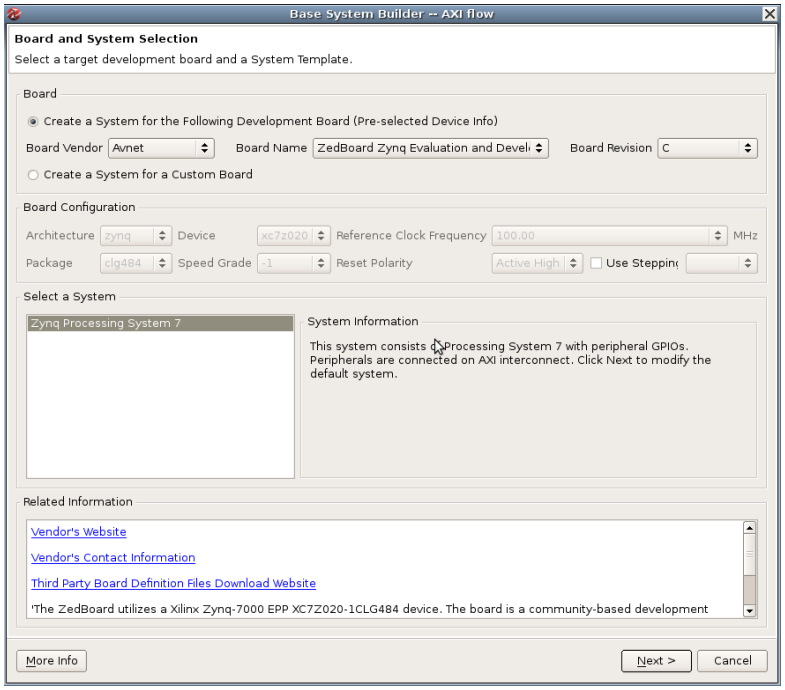


Ilustración 20: Especificación placa de desarrollo en XPS

Finalmente para terminar la creación del diseño hardware se configurarán y **se especificarán** los **periféricos** comunes que se quieren añadir. Además de **añadir** y poder **modificar** los diferentes periféricos se pueden activar las interrupciones para dichos periféricos.

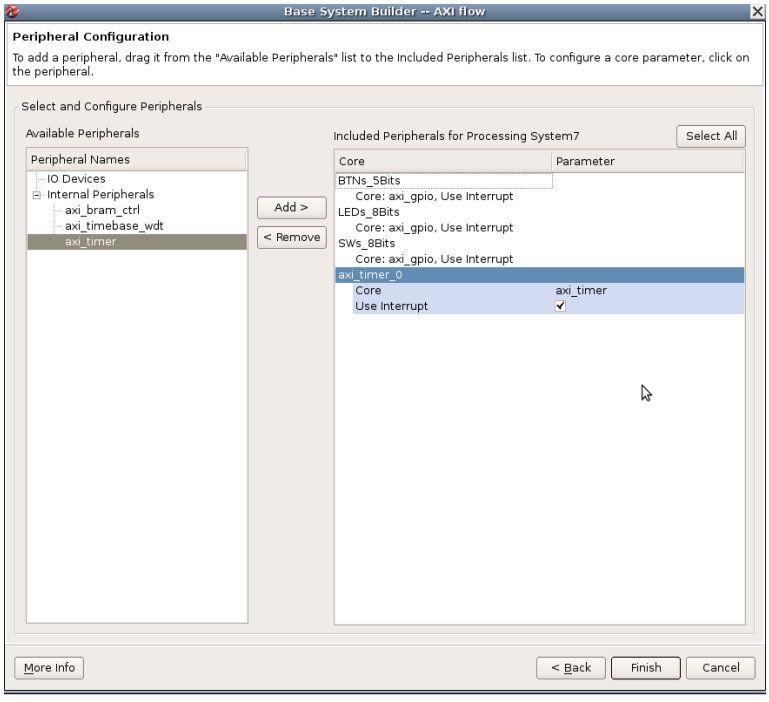


Ilustración 21: Gestión de periféricos de un diseño hardware en XPS

Una vez introducidos los periféricos e interrupciones se finalizará el proceso para la realización de la plantilla del diseño del sistema. Al finalizar saldrá la **pantalla principal del XPS**, desde la cual se puede **configurar** y realizar **modificaciones** al diseño hardware, y **añadir** y **editar** uno o varios **IPs**, y gestionar los puertos y las direcciones de memoria.

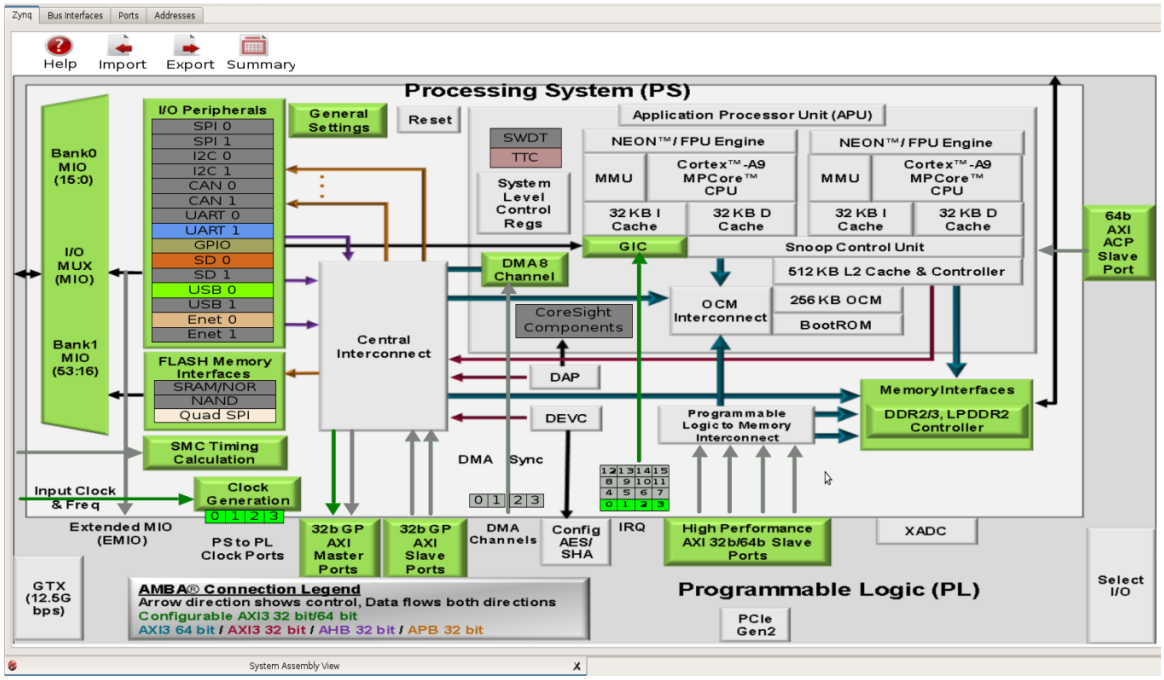


Ilustración 22: Pestaña *System Asembly View* en XPS

Una vez completado la creación del diseño de sistema general pasaremos a la creación de la estructura del periférico.

5.1.1 Creación del periférico

En esta fase explicaremos el proceso realizado para la **creación** del **proyecto HW** del periférico y la creación de una plantilla sobre la que basarse para la implementación del periférico. Además se realizarán configuraciones sobre el componente para la gestión de memoria utilizado.

En primer lugar, partiendo del proyecto del sistema implementado desde la herramienta **XPS** en la fase anterior, daremos a la opción de la creación o importación de un nuevo IP, e indicaremos que lo que queremos es **crear** las **plantillas** para la implementación un nuevo periférico. Para ello utilizaremos el asistente para creación e importación de periféricos.

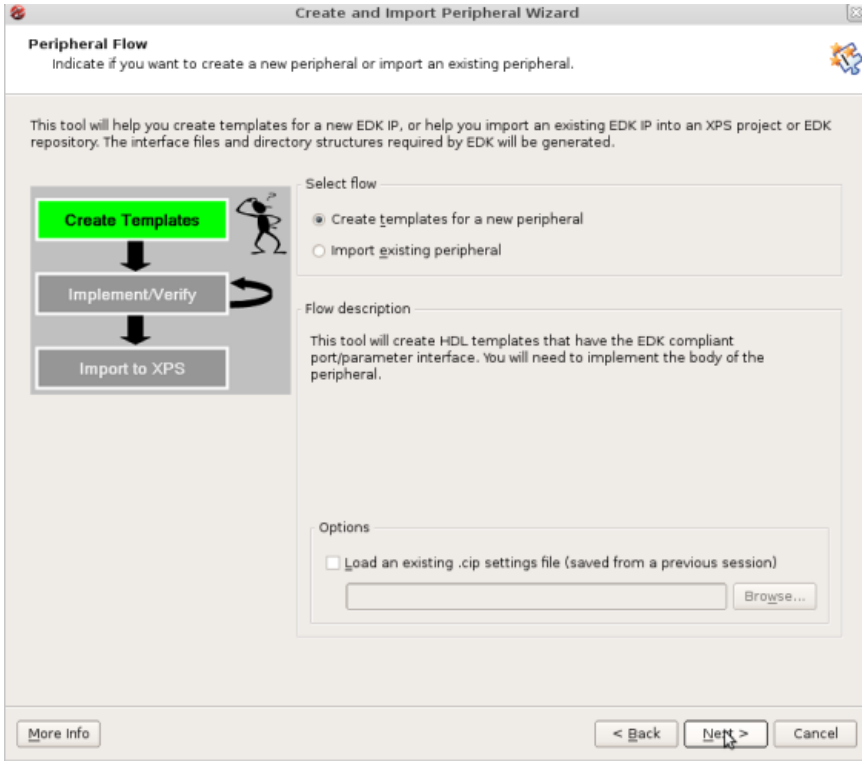


Ilustración 23 : Creación Periférico XPS

Posteriormente nos dejará a elegir si que quiere crear el periférico en la **ruta** del **proyecto** XPS o crear una nueva ruta donde crear un proyecto completo para la herramienta ISE. En nuestro caso hemos elegido por comodidad introducirlo dentro del proyecto XPS, pero es una decisión que sólo afecta a las rutas donde se ubican los ficheros del periférico, y no es relevante para el resultado final.

Una vez elegida la ruta de su ubicación, se pasará a poner un **nombre** al periférico, así como una **versión** del mismo, esto permite un ligero control de las versiones del periférico.

La primera decisión que tenemos que realizar a la hora de crear la plantilla es elegir el tipo de interconexión, como se ha comentado en el apartado de diseño del periférico, el periférico diseñado utilizará el protocolo de comunicaciones AXI4Lite, el protocolo de comunicación de la especificación AMBA más sencillo. Este tipo de interconexión y el componente IPIF correspondiente permite un acceso y gestión de datos mediante registros.

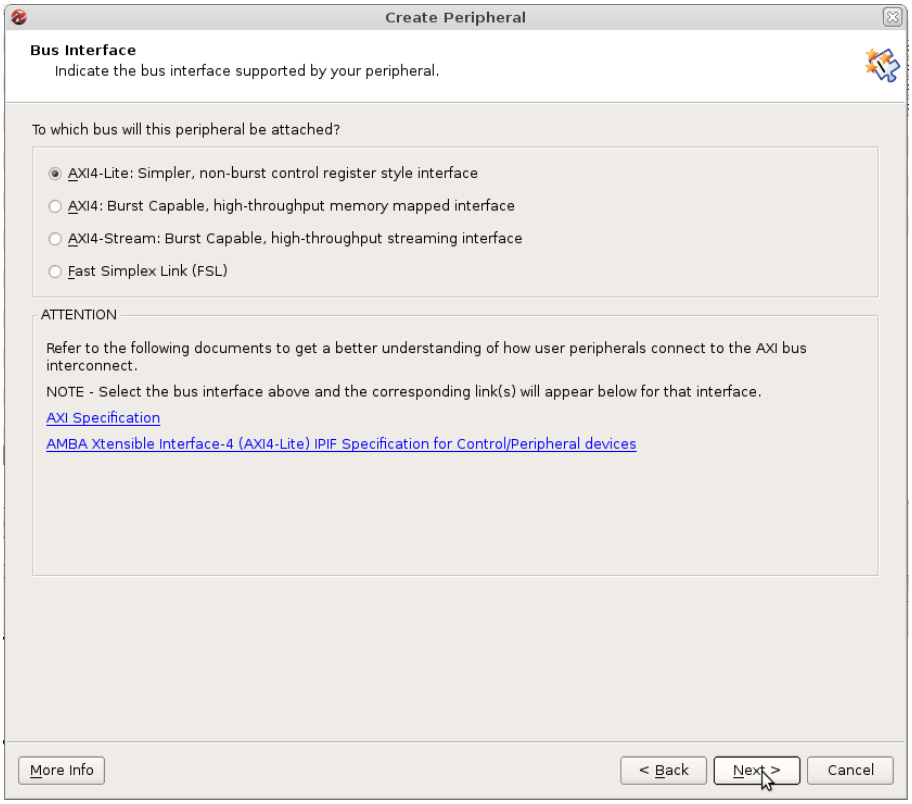


Ilustración 24: Elección interconexión periférico

A continuación pasaremos a **configurar** la forma y el contenido el componente encargado de los **servicios** para la **interfaz para el IP**, el Axi4Lite IPIF. En esta parte podremos elegir el tipo de periférico y de funcionalidades que se crearán en la plantilla.

Para comenzar esta configuración, especificaremos la configuración general del módulo según nuestro diseño. En primer lugar **no** marcamos la opción de crear un periférico con soporte para las interfaces de maestro de AXI4Lite, debido a que el dispositivo solo va a utilizar algunas de las interfaces del protocolo de **esclavo**.

Posteriormente, especificamos los **servicios** del funcionamiento como periférico esclavo, tales como la implementación en la plantilla del ***reset*** vía **software**, el uso de **registros** en la lógica de usuario y la inclusión de un ***timer*** para los datos.

El *phase timer* se encarga de gestionar los **ciclos** que espera la interfaz del IP cuando realiza peticiones de escritura/lectura al IP - lógica de usuario. Esto se realiza para que el módulo IPIF solo espere un tiempo concreto a que el IP responda a una petición de lectura/escritura, evitando que se quede bloqueado esperando al IP

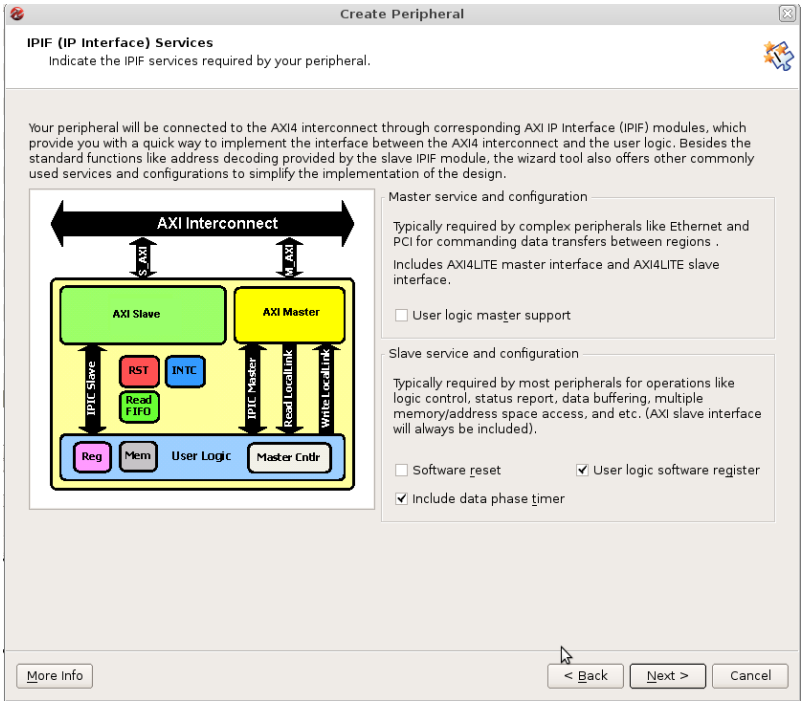


Ilustración 25: Configuración IPIF XPS

Una vez explicada la configuración general del IPIF se pasará a configurar su funcionamiento principal. En primer lugar se configurará el **número de registros** que se van a utilizar en el periférico. Esto es totalmente necesario para configurar las señales de **selección de los registros** conocidas como ***Chip Enable***, como se explica en el asistente de creación de la plantilla. En nuestro caso utilizaremos tres registros, dos para almacenar los sumandos y otro para el resultado de la suma.

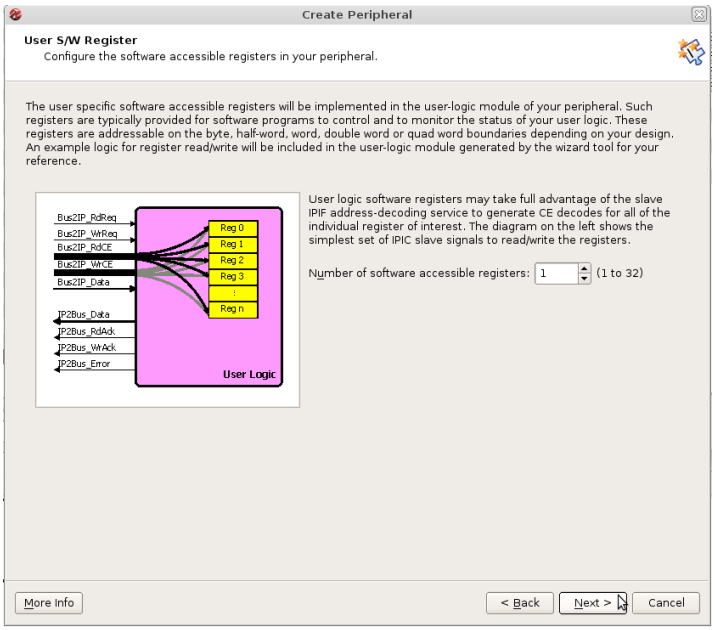


Ilustración 26: Selección de número de registros del periférico

Una vez escogidos el número de registros del periférico, seguiremos configurando el IPIF. A continuación elegiremos las **interfaces** que ofrece el módulo a la lógica de usuario, es decir, definiremos las interfaces que contendrá nuestra lógica de usuario y por donde recibirá los datos del módulo IPIF.

En nuestro caso hemos elegido que se utilicen las siguientes **interfaces** necesarias para la lógica de usuario, indicadas y especificadas en el diseño del periférico. Como se comentó en el diseño del periférico, debido a que se va a utilizar una **selección de registros** mediante el ***chip enable*** de una manera simple y eficiente, no se necesitarán las interfaces de selección de dirección de memoria "Bus2IP_Addr", y la señal de selección de chip "Bus2IP_CS". Igualmente tampoco se utilizará la señal para seleccionar el proceso que se va a realizar escritura o lectura "Bus2IP_RNW", debido a que siempre la **escritura** en la lógica de usuario tiene más **preferencia** que la **lectura.**

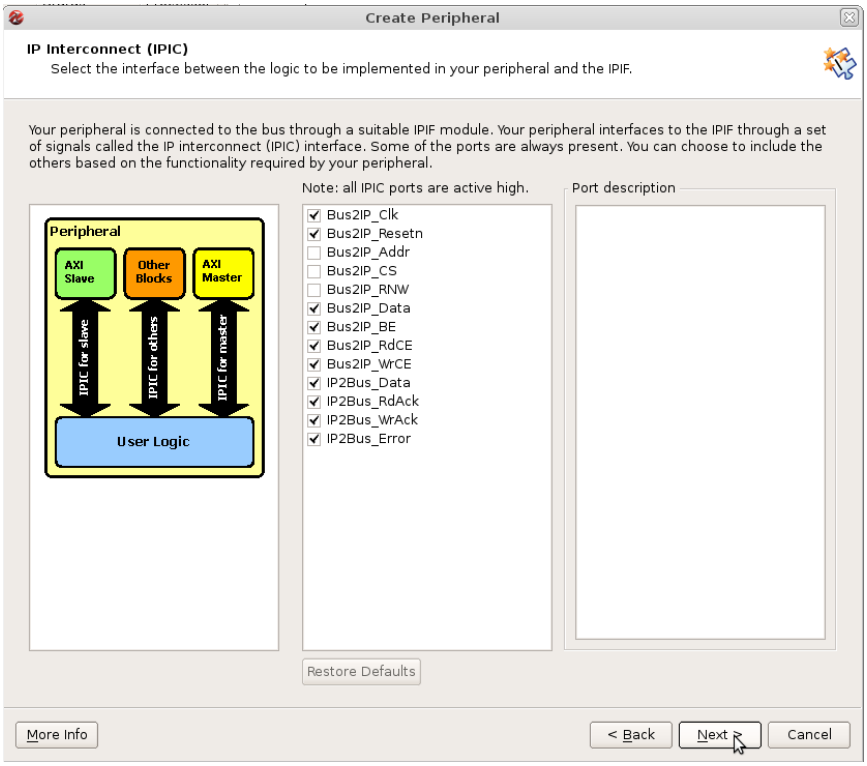


Ilustración 27: Selección interfaces de lógica de usuario

Una vez definida la configuración del módulo IPIF, se especificarán añadidos opcionales que se pueden elegir. En primer lugar el asistente permite crear una **plataforma de simulación** para el periférico.

Esta plataforma se encarga de generar ficheros ***testbench*** como plantillas para que funcionen en esta plataforma de simulación. Se ha escogido no utilizar esta plataforma de simulación y se ha escogido realizar los ***testbench* propios**. Esto es debido a que se prefiere tener el control sobre todo lo que se quiere probar en el periférico sin ninguna plataforma por medio utilizando pruebas unitarias de simulación, en lugar de realizar una simulación del sistema en sí mediante una plataforma de simulación, no demasiado útil para un periférico con una lógica simple.

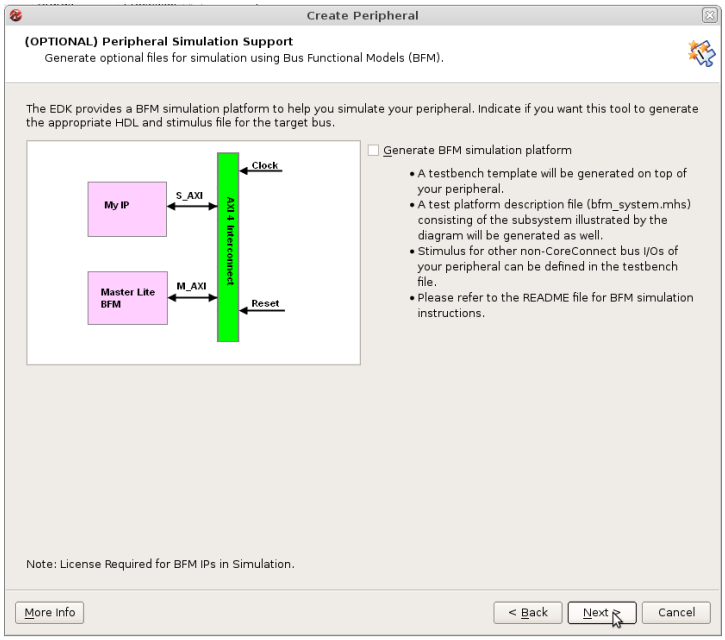


Ilustración 28: Selección plataforma simulación en periférico

Finalmente elegiremos otras opciones para la creación de la plantilla del proyecto. En nuestro caso hemos elegido que la lógica de usuario se realice en **VHDL**, debido a ser el lenguaje de descripción de hardware más adaptado al desarrollador hardware.

Para concluir se ha decidido la generación de los ficheros para facilitar el proceso de **síntesis** del sistema en ISE y que se genere la **plantilla** para un **driver** capaz de comunicarse vía software con las interfaces del periférico.

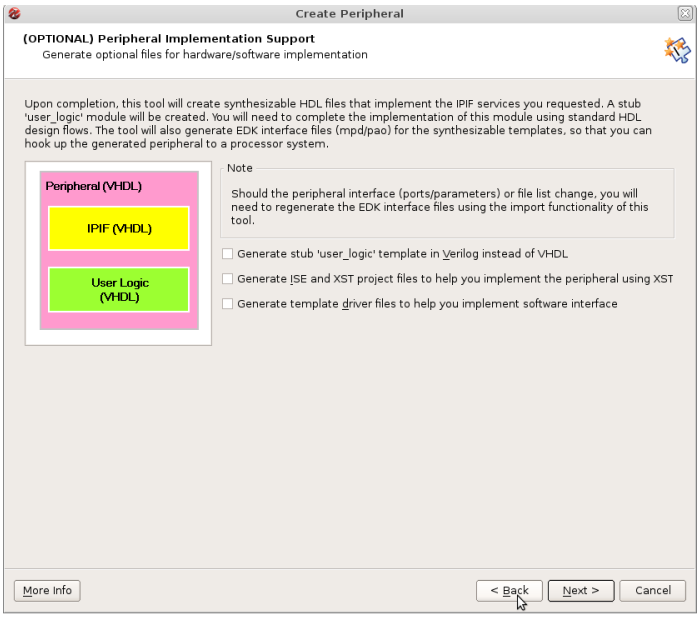


Ilustración 29: Características en la creación de la plantilla del periférico

5.1.2 Implementación del periférico

En este apartado comentaremos como se ha realizado la implementación del periférico utilizando la herramienta ISE. Partiendo del proyecto de plantilla creado anteriormente. En primer lugar ejecutaremos la herramienta de desarrollo hardware **ISE** de la suite, donde se realizará la implementación del periférico.

Una vez ejecutada la herramienta, pasaremos a **abrir** la **plantilla** del proyecto realizado. Para ello dependiendo la opción de la **ruta** de creación en el anterior paso, buscaremos en la ruta del proyecto XPS o en una ruta nueva como un proyecto de ISE. En los dos casos buscaremos el fichero con el mismo nombre que el periférico y de extensión **".xise"**, correspondiente a la plantilla del periférico.

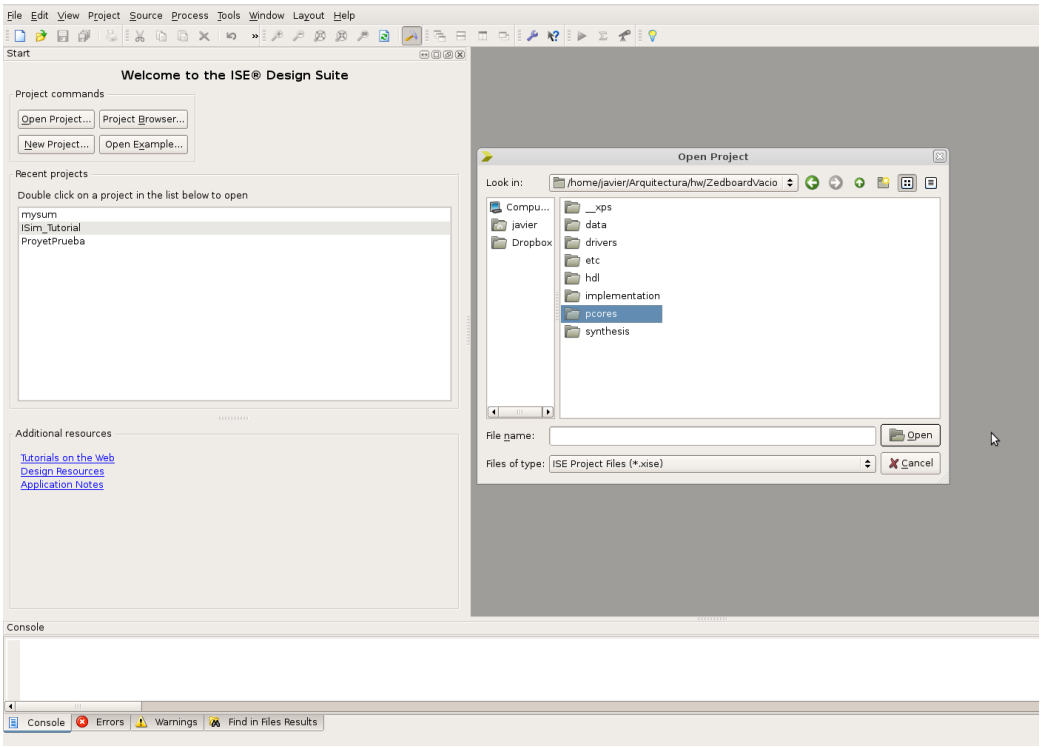


Ilustración 30: Abrir proyecto plantilla del periférico

Una vez abierto el proyecto estaremos en la pantalla principal de la herramienta ISE. El programa está distribuido por tres secciones principalmente, la sección principal donde podemos navegar con las pestañas **diseño**, **ficheros** y **librerías**, la ventana de **código** y la sección de **consola** y **errores**. En la parte de diseño podemos observar la **jerarquía** de implementación y simulación, con todos los ficheros del proyecto, y en la parte de parte de abajo podemos observar las acciones que se pueden realizar sobre el fichero escogido.

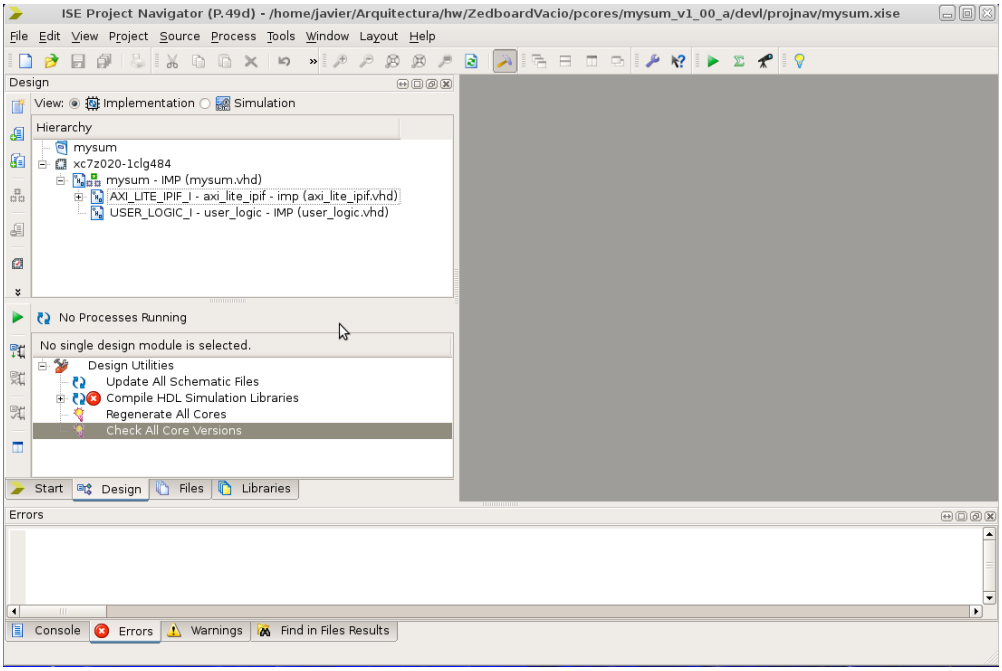


Ilustración 31: Ventana principal periférico en ISE

En primer lugar comprobaremos que la plataforma del hardware del sistema es el correcto, para ello seleccionaremos la **plataforma hardware** y comprobaremos que está correcta, examinando las **propiedades de diseño**. Comprobamos que el diseño de más alto nivel se va a describir mediante un HDL, que el hardware equivale al del sistema y que el lenguaje utilizado por defecto sea VHDL, el que va a utilizar el desarrollador hardware.

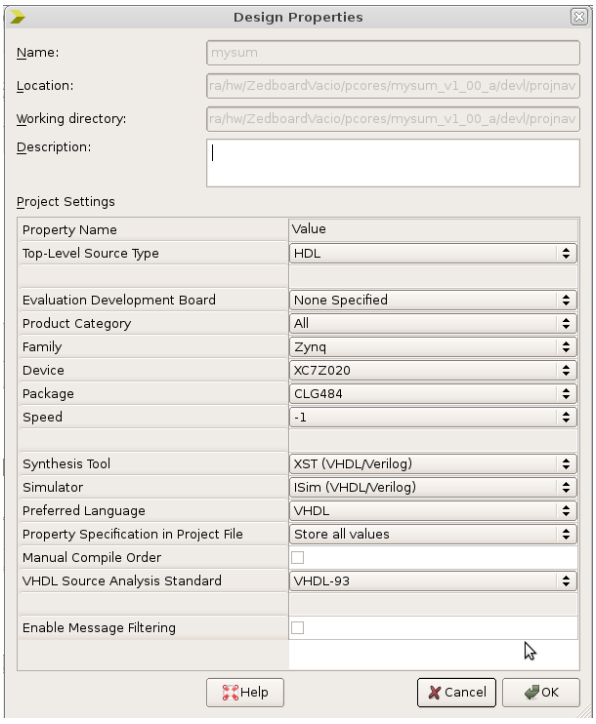


Ilustración 32: Propiedades de diseño de la plataforma hardware

Para realizar la **implementación** de la **lógica de usuario**, vamos a realizar el diseño de la lógica de usuario escribiéndola en VHDL y partiendo de la plantilla creada por el proyecto. Una vez terminada la implementación pasaremos a la **compilación** y **comprobación** del código VHDL escrito.

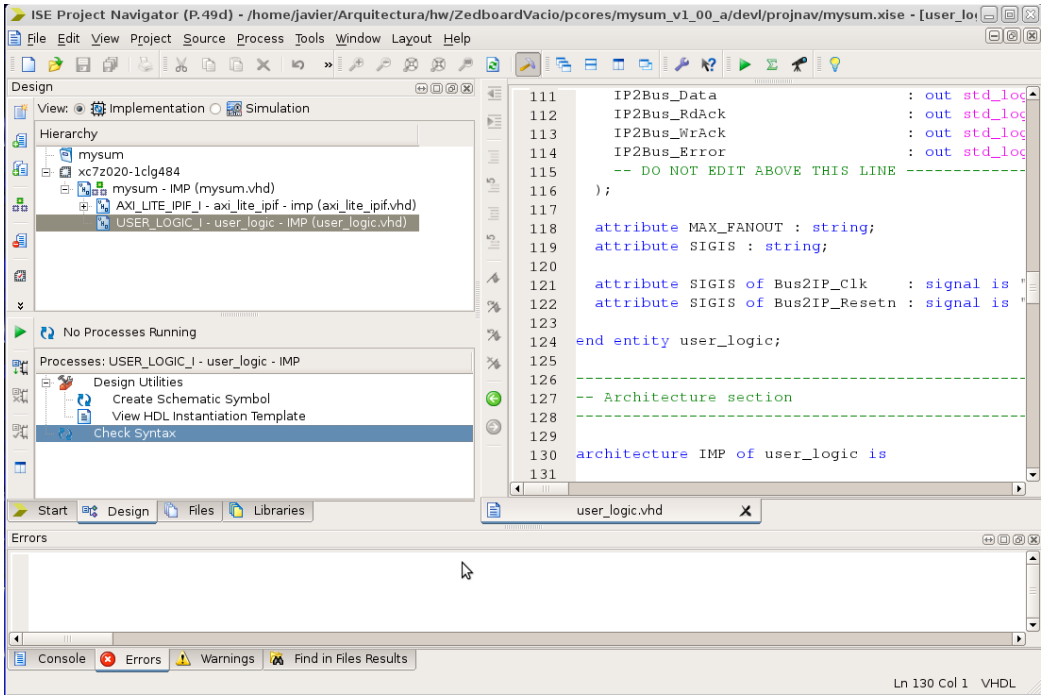


Ilustración 33: Chequeo sintaxis código VHDL en ISE

Una vez comprobado que la sintaxis y semántica de la lógica del usuario está totalmente correcta, pasaremos a **compilar** y comprobar la sintaxis del **componente de más alto nivel**, correspondiente al componente Axi4lite IPIF en nuestro caso llamado como el periférico "mysum.vhd". Para ello como en el caso anterior realizaremos el comando "check syntax" para su comprobación.

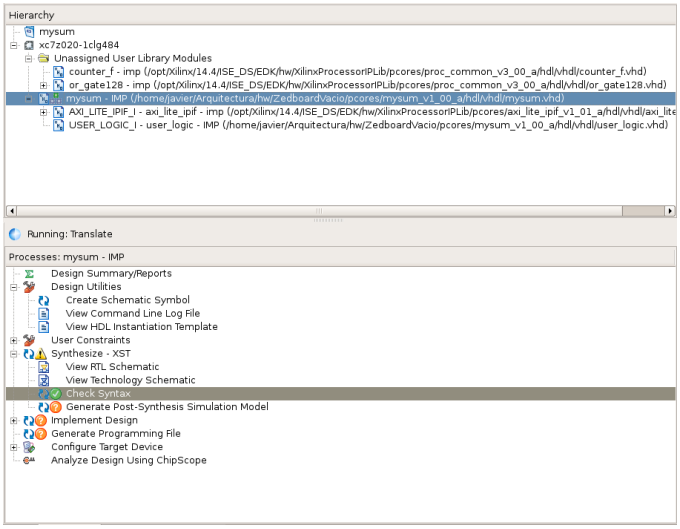


Ilustración 34: Compilación componente de más alto nivel VHDL en ISE

En el caso de que haya algún **error** o advertencia con el chequeo y compilación del código VHDL, se mostrarán los errores del código en la ventana de errores, indicando la línea del código y el tipo de error.

En el caso de querer una descripción más detallada de todos los mensajes del sistema, se puede mostrar en una pestaña utilizando la pestaña de resumen y notificaciones del sistema.

En esta pestaña se pueden comprobar todos los mensajes de **error y advertencias** del **proyecto** de las fases del diseño hardware (compilación, síntesis, mapeo, exportación o simulación) de una manera **detallada**, así como los errores de sus herramientas logiCore o su simulador iSim. Además también se puede consultar información del diseño hardware realizado en modo de **resumen**.

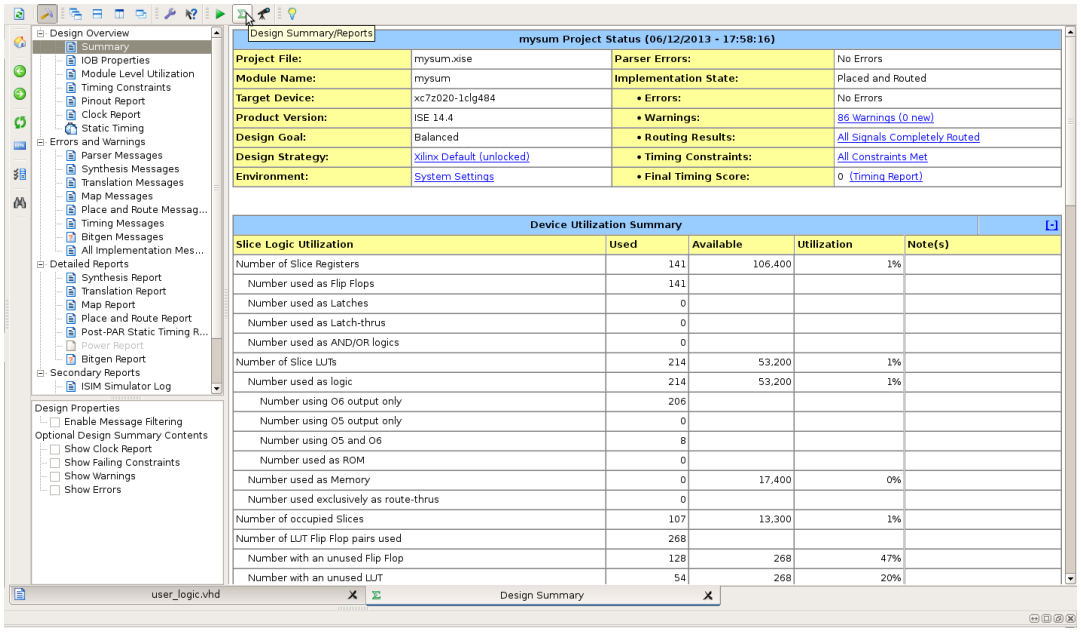


Ilustración 35: Mensajes y resumen de diseño

Una vez comprobado que todo el código está correctamente compilado pasaremos a recompilar el sistema completo y generar la **síntesis** y posteriormente implementarlo realizando su ***place and route*** del sistema.

La **descripción** de los diferentes procesos que se tienen seguir para la **generación** e implementación del diseño hardware son las siguientes:

- **Síntesis**: Este proceso se encarga de realizar la compilación de todo el sistema completo, y se encarga de generar la ***netlist***, que conecta puertas lógicas y flip-flops juntos. Para la realización de esta síntesis, la herramienta ISE utiliza la herramienta de síntesis provista por Xilinx, la herramienta **XST**, que genera un fichero de extensión "prj".
- ***Place and Route*:** El proceso de *place and route* son varios procesos que se encargan de comprobar los elementos de la *netlist*, y mapearlos a recursos físicos de la FPGA. Este proceso engloba el proceso de traducción del *netlist* y del mapeo a recursos físicos de la FPGA.

Para permitir la correcta **importación del periférico** creado en el sistema completo, sólo es **necesario** generar la *netlist* y realizar la **síntesis** del sistema. Pero adicionalmente, a pesar de no ser necesario, se realizará la implementación y el *place and route* del diseño para comprobar que el sistema se podría realizar correctamente, comprobando su correcta composición.

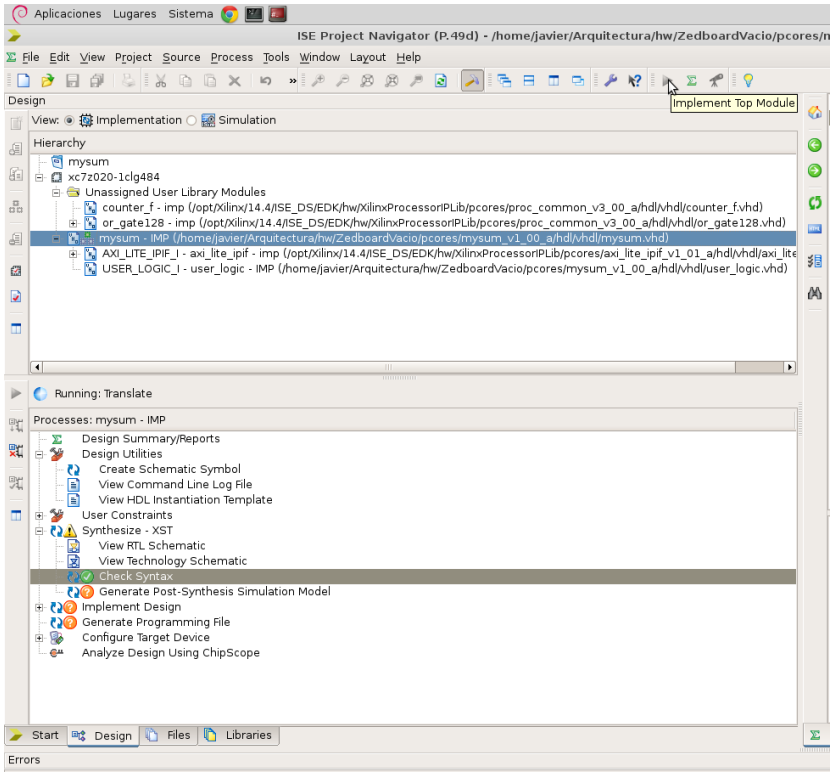


Ilustración 36: Síntesis e implementación del diseño hw del periférico

Una vez terminada la implementación del diseño hardware del periférico se pasaran a realizar las **pruebas unitarias** sobre el periférico para comprobar el funcionamiento del hardware antes de su importación, estas pruebas unitarias se realizarán mediante *tesbench*, utilizando el simulador. El desarrollo y los resultados de las pruebas unitarias se explicarán en el apartado **¡Error! No se encuentra el origen de la referencia.**.

5.1.3 Importación periférico

Una vez se ha obtenido el **diseño hardware del periférico sintetizado** e **implementado**, y después de comprobar su correcto funcionamiento a través de las **pruebas** unitarias del **simulador**, se pasará a importar el periférico en el resto del sistema. Para ello, desde el proyecto del sistema realizado en XPS **importaremos el periférico,** utilizando la herramienta para la creación e importación de periféricos.

Desde la herramienta de creación e importación de periféricos elegiremos en este caso que se quiere realizar la importación de un periférico. Una vez seleccionado el programa pide que indiquemos el tipo de ficheros que definen el periférico. En nuestro caso señalaremos todos los tipos de ficheros, debido a que se contienen **ficheros de descripción de hardware** o **HDL**, ficheros ***netlist*** generados durante la **síntesis** del periférico y los ficheros de **documentación** que han introducido en el proyecto del periférico, las especificaciones del módulo IPIF, la especificación AMBA en general y AXI en particular y una breve descripción del funcionamiento de la lógica de usuario.

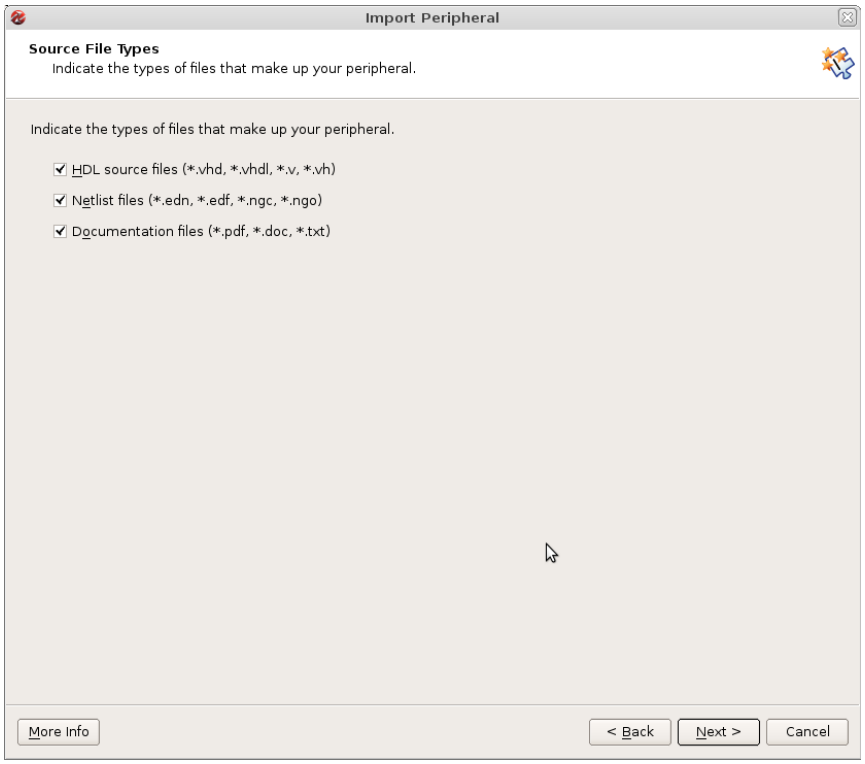


Ilustración 37: Selección de tipos de ficheros a importar del periférico

A continuación indicaremos el **nombre** y la **versión** del periférico a importar, en nuestro caso como queremos sustituir el periférico que creamos al principio, le daremos el mismo nombre y la misma versión, para sustituirlo.

Una vez elegido el nombre y versión, el programa indicará de qué manera vamos a buscar los ficheros HDL que describen el periférico. Existen varias maneras de **importar** los **ficheros HDL**:

- Utilizando los datos de una **importación anterior** mediante un fichero de extensión ".mpd".
- Partiendo de un fichero con extensión ".prj" resultado de realizar la **síntesis** un proyecto ISE con la herramienta de síntesis de Xilinx **XST**. Mediante esta opción también se encargará de **importar** las **librerías** necesarias para la compilación del código HDL.
- Utilizando un fichero que indica el **orden de compilación** de ficheros HDL y las **librerías** que se importan mediante un fichero PAO.
- Buscando manualmente los ficheros HDL y las librerías necesarias para su compilación.

En nuestro caso vamos a seleccionar los ficheros HDL y las librerías necesarias mediante un fichero ".prj" obtenido al realizar la síntesis del proyecto de diseño HW del periférico.

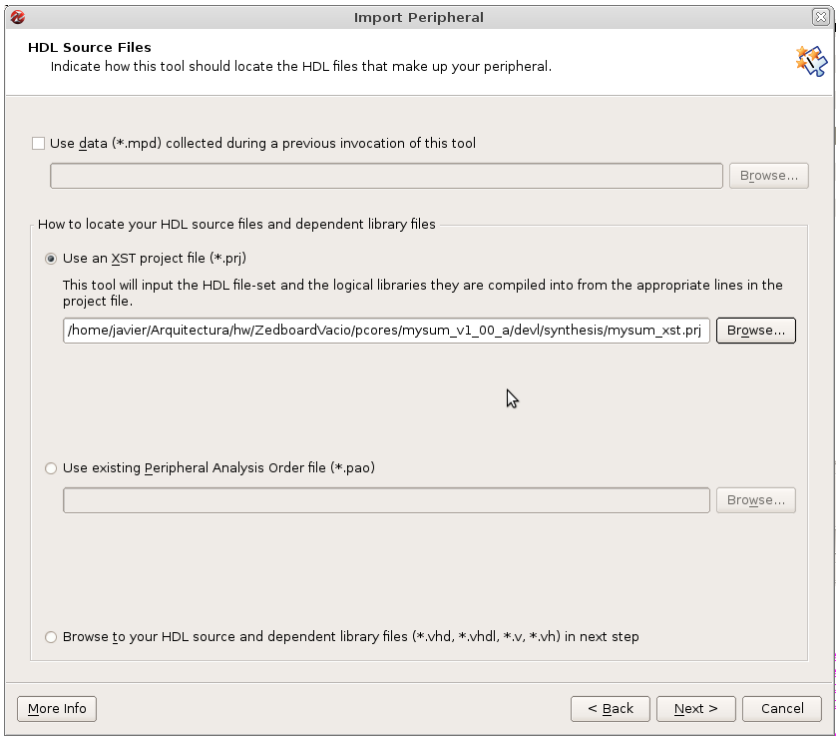


Ilustración 38: Selección tipo de importación de ficheros HDL

Una vez indicado el modo de importar los ficheros HDL, se mostrarán todos los ficheros HDL que se han obtenido mediante algunos de los ficheros comentados anteriormente. Adicionalmente en esta sección podremos añadir más ficheros o librerías en el caso de ser necesarias.

En este apartado se pueden observar que se importan correctamente los dos ficheros **VHDL** correspondientes a la implementación del **módulo IPIF**, y a la **lógica de usuario**, que se encuentran organizados en la **librería lógica** con el mismo nombre que el periférico "mysum_v1_00.a". Por otro lado podemos comprobar que se han importado las librerías del "axi_lite_ipif" y las librerías generales necesarias.

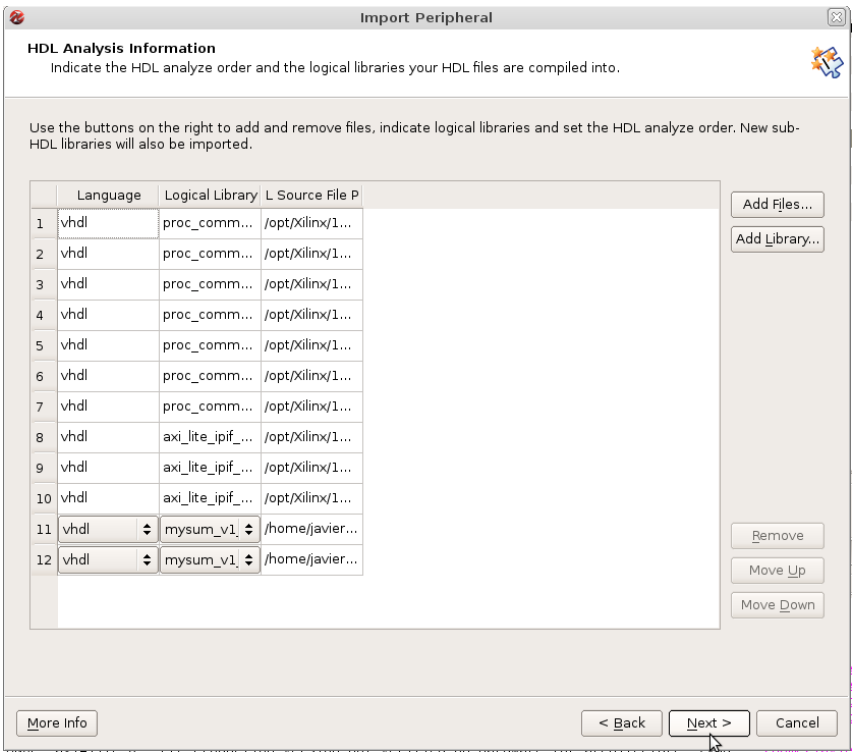


Ilustración 39: Selección ficheros HDL para importación del periférico

Una vez se hayan elegido todos los ficheros HDL y librerías que componen el periférico a importar, se pasará a elegir el tipo de **intercomunicación** que se quiere tener entre el periférico y el resto del sistema. Como se ha comentado en el diseño del periférico, el periférico se conectará con el resto del sistema por el **bus** S_AXI del sistema, y mediante las interfaces definidas por la especificación **AXI4Lite**, tomando el rol de componente **esclavo**.

Una vez elegido el tipo de intercomunicación entre el periférico y el sistema, en la siguiente pantalla del asistente aparecerá la **relación** entre los **puertos** del **protocolo** de intercomunicación y los puertos del **periférico**. En nuestro caso, las interfaces del periférico corresponden a las que define el fichero **HDL** de **mayor nivel**, es decir, nuestra implementación del módulo **IPIF** llamado como el periférico.

Por norma general, el mapeo entre los puertos se realiza manualmente, pero como se han respetado la **nomenclatura** para nombrar a los **puertos** en el periférico, el sistema ha realizado la relación **automáticamente**. Aun así, se ha comprobado que está **correctamente** realizado el mapeo y es correcta la **correlación** entre las interfaces del periférico y las definidas en el AXI4Lite como esclavo.

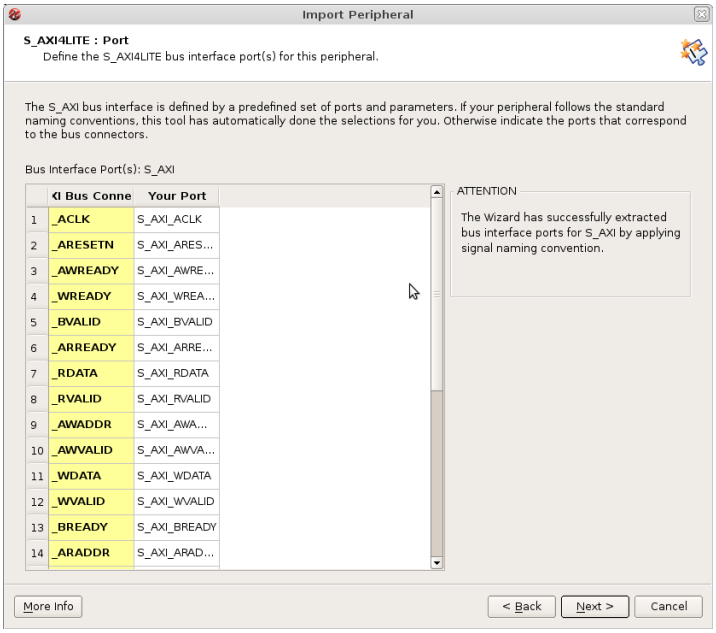


Ilustración 40: Relación de puertos entre bus S_AXI y periférico

Una vez configurada la interconexión entre el periférico y el sistema, se configurarán los parámetros del periférico definidos como **variables generales** del componente HDL de mayor nivel.

En primer lugar se elegirá el espacio de **memoria** asignado para el acceso al periférico. Esto se puede realizar de varias formas, mediante **variables generales** definidas en el fichero HDL, o asignándole un rango de memoria **manualmente**. Nosotros usaremos las variables generales "C_BASEADDR" y "C_HIGHADDR", para definir la **dirección base** y la **dirección alta** del rango de memoria.

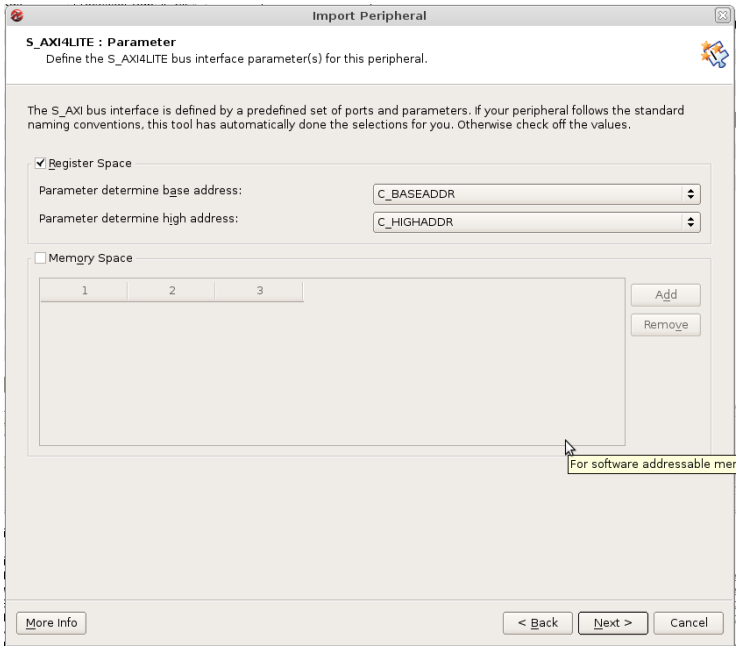


Ilustración 41: Asignación parámetro de rango de memoria

Una vez asignados los parámetros que definen el rango de memoria que utilizará el periférico, se configurarán los valores del resto de parámetros del periférico, definidos igualmente mediante variables generales en el fichero HDL de mayor nivel.

En la siguiente ventana podremos examinar todos los parámetros que contiene el periférico, consultar su **nombre**, **tipo** y **valor por defecto**, así como otros atributos avanzados del parámetro.

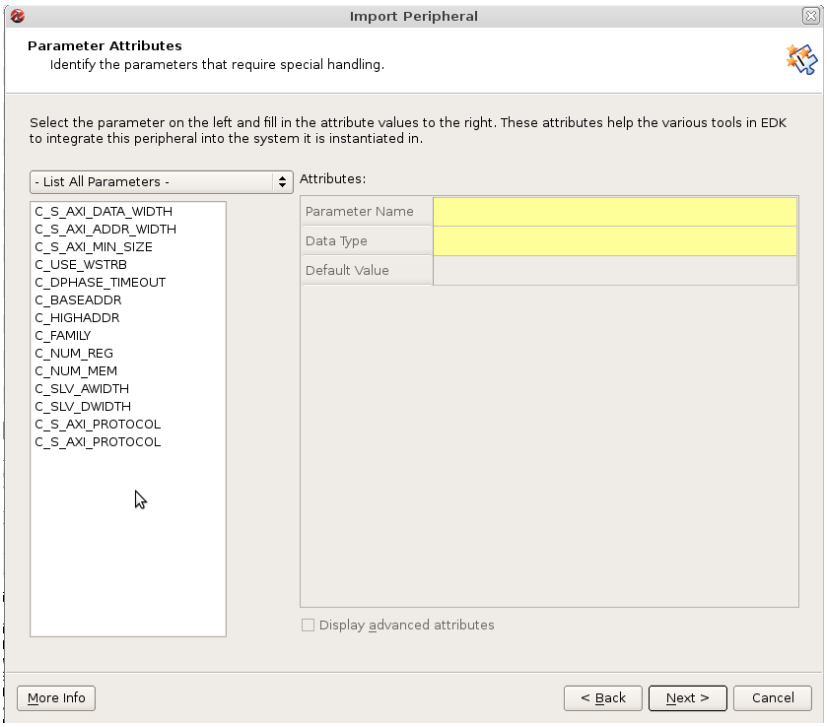


Ilustración 42: Consulta de atributos del periférico a importar

Una vez consultados que están todos los parámetros que se desean se pasará a comprobar la **lista de puertos** que contiene el periférico. En este apartado se pueden revisar todos los atributos de los puertos que tiene el periférico como su **nombre**, si la dirección es de **entrada** o **salida**, **conexión por defecto** del puerto al respectivo puerto del bus del sistema del protocolo, y la **dimensión** del puerto, en el caso de que sea **unidimensional** el campo estará vacío.

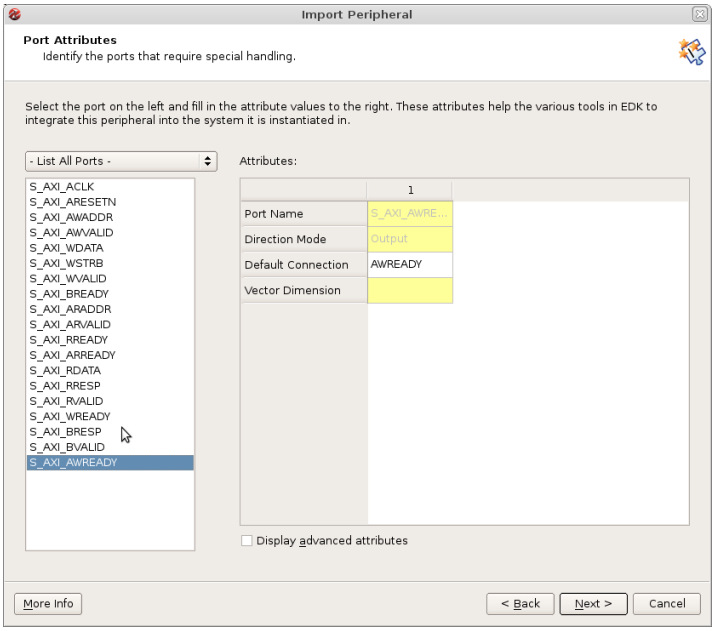


Ilustración 43: Consulta de puertos del periférico a importar

Al terminar de consultar todos los aspectos del periférico se terminará su generación.

Una vez importado correctamente en el sistema como un IP, se podrá introducir en el sistema fácilmente desde el catálogo de IP del sistema. Después de confirmar que se quiere añadir el IP al sistema, la herramienta mostrará el periférico creado con el **formato** de la herramienta **XPS**. Desde esta ventana podremos **modificar** los **atributos** definidos y realizar su **configuración**. Se podrá tener acceso a esta ventana también después su implantación por si se quiere cambiar algún aspecto de la después de implantación en el sistema.

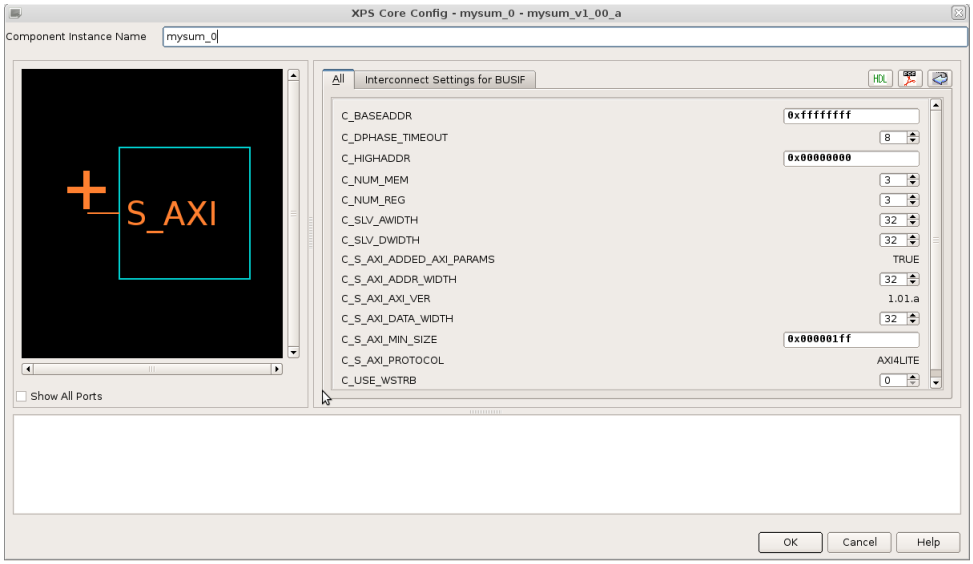


Ilustración 44: Ventana de configuración del periférico implementado

Una vez implementado correctamente el periférico en el sistema se comprobará que se ha conectado correctamente al **bus** del sistema correspondiente. Para ello iremos a la pestaña de "Bus Interfaces" donde se especifica la interfaz del **bus** al que están conectados todos los periféricos y el **rol** de cada periférico. Como se puede observar en la siguiente captura de pantalla, el periférico está **interconectado** a la interfaz esclava del bus con nombre **"axi4lite_0"**.

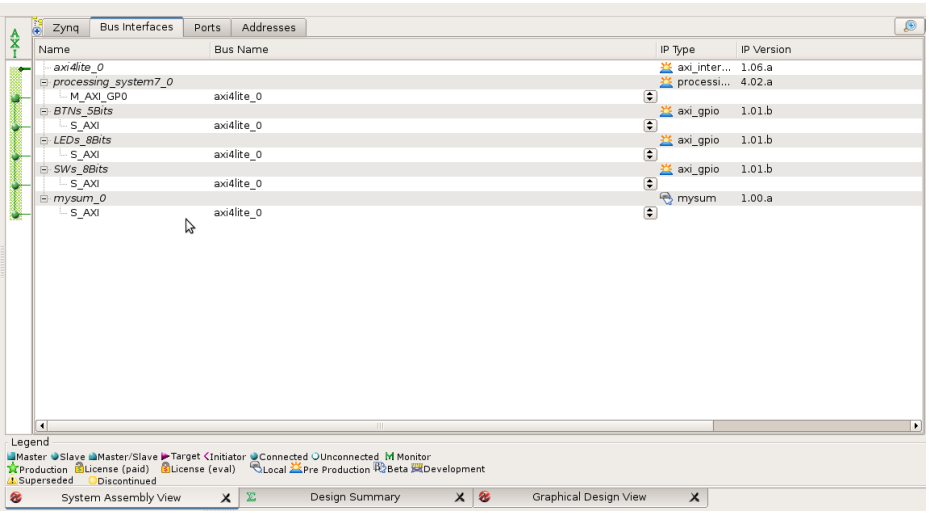


Ilustración 45: Lista de interfaces de los buses del sistema

Finalmente asignaremos un rango de memorias al periférico, mediante la pestaña "addresses". La herramienta dará un valor a los **parámetros** "C_BASEADDR" Y "C_HIGHADDR" del periférico para que funcione a partir de ese rango de **direcciones**. Para ello declaremos que se necesitarán 4Kb de memoria para el periférico como se comentó en el apartado de diseño. Generaremos **direcciones aleatorias** para todos los dispositivos del sistema, para que la herramienta se encargue de **ordenar** y **optimizar** las diferentes **direcciones de memoria**.

Como se puede observar en la siguiente captura de pantalla las direcciones de memoria que se le asignan son las definidas en el apartado del diseño del sistema.

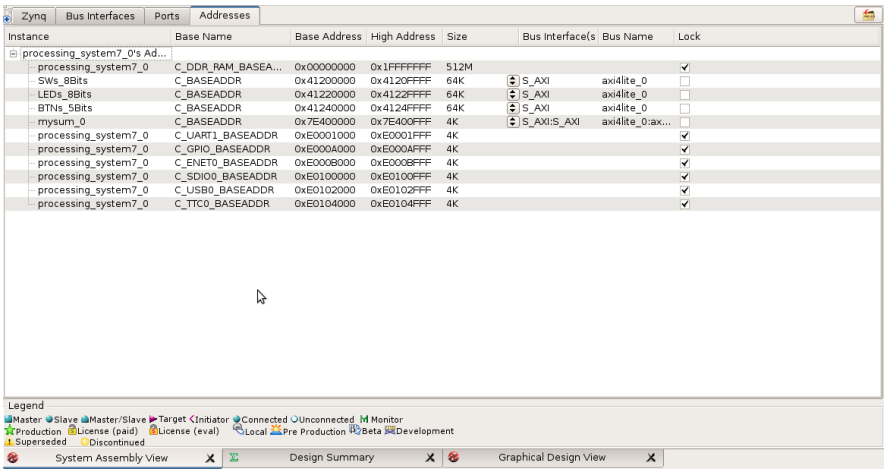


Ilustración 46: Direcciones de memoria de todos los componentes del sistema

5.1.4 Exportación diseño hardware

Una vez se cuente con el diseño hardware el siguiente paso es **exportarlo** a un **fichero**, para poder programarlo en la placa o utilizarlo en el XSDK. El fichero donde se exporta el diseño hardware se conoce como **bitstream**, y tiene extensión **".bit".**

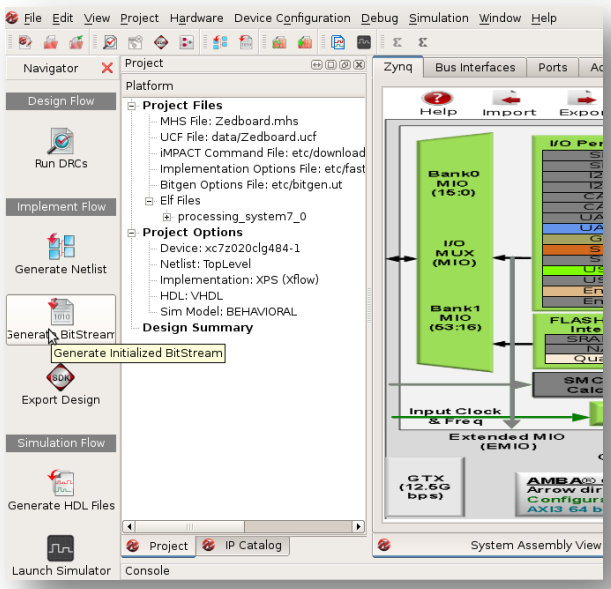


Ilustración 47: Generación de *bitstream* en XPS

Para **generar** el fichero **bitstream**, simplemente se tiene que dar al respectivo comando en el XPS. Después de un **tiempo** dependiente de la potencia del procesador, el programa generará un fichero llamado **"System.bit".**

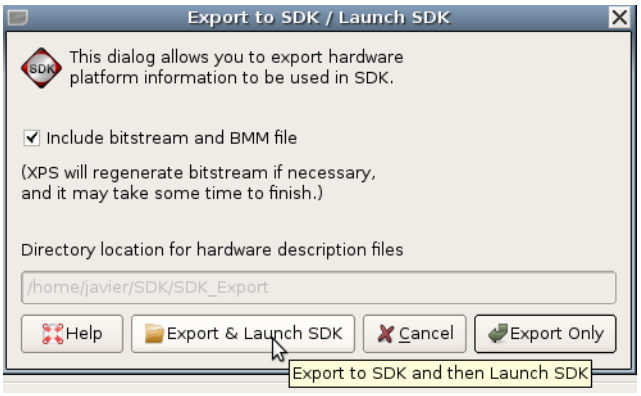


Ilustración 48: Exportación de diseño HW desde XPS a XSDK

Este bitstream puede utilizarse para ser **importado** desde otros programas. Para exportar el código para el XSDK simplemente se presionará el botón, especificando que tiene que **exportar** y **abrir** el **XSDK**. En ese momento se arrancará el XSDK, y se cargará el diseño hardware en el espacio de trabajo que seleccionemos.

5.2 Implementación y compilación de código

La **implementación** y **compilación** del código se ha realizado íntegramente en el **XSDK**. Al iniciar el programa éste preguntará qué **directorio** queremos como **espacio del trabajo**, dónde se almacenarán las aplicaciones, los BSP y los diseños hardware importados.

En primer lugar se importará el diseño hardware realizado anteriormente, utilizando la exportación de XPS. Esta exportación creará un proyecto hardware en el XSDK, que se utilizará para poder asignarse a otros proyectos de aplicaciones.

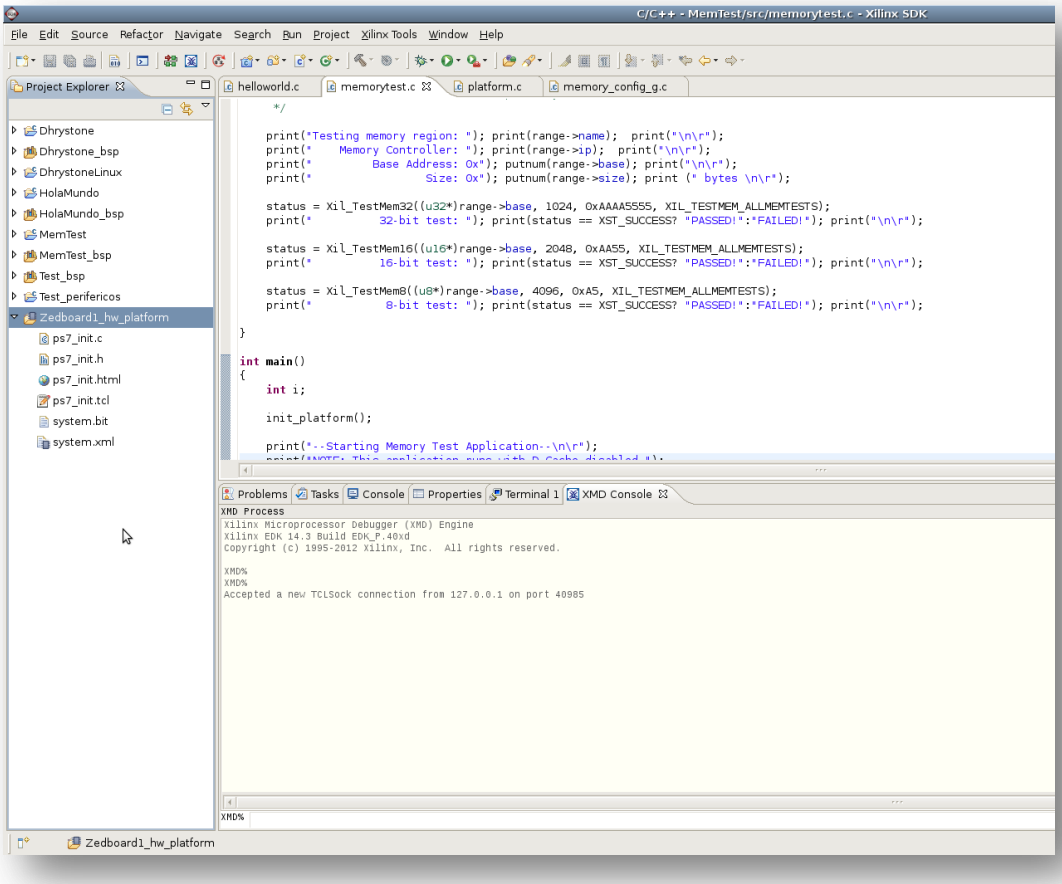


Ilustración 49: Especificación de diseño HW en XSDK

El proyecto hardware está formado por un pequeño código que se encarga de **inicializar** el **hardware** cuando se ejecuta código, el "System.bit", y documentos que contienen una **descripción** de la inicialización.

Las aplicaciones para su ejecución necesitan un **BSP**, que contiene todos los **componentes software** necesarios para adaptarse a un sistema operativo y su entorno, aunque sea una aplicación sobre el **hardware** o ***standalone***.

Los paquetes de soporte de la placa o BSP, pueden ser creados a la vez que se crea la aplicación, o de una manera independiente asociando posteriormente la aplicación a ese BSP.

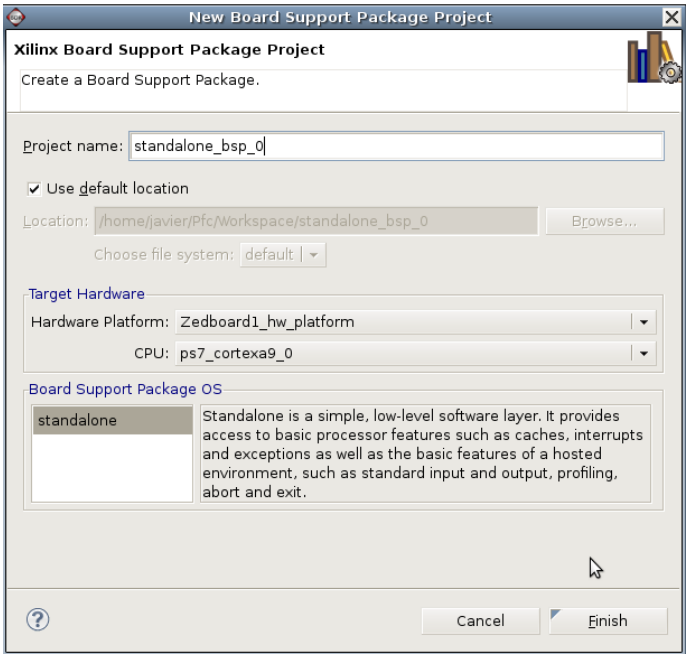


Ilustración 50: Creación de proyecto BSP en XSDK

Normalmente al crear un BSP manualmente permite más opciones a la hora de elegir un sistema operativo.

Para **implementar** las **pruebas** se han creado varias aplicaciones, una para cada prueba. Para facilitar el proceso de creación, se han utilizado **plantillas** de proyectos de aplicaciones, generadas automáticamente por el kit de desarrollo.

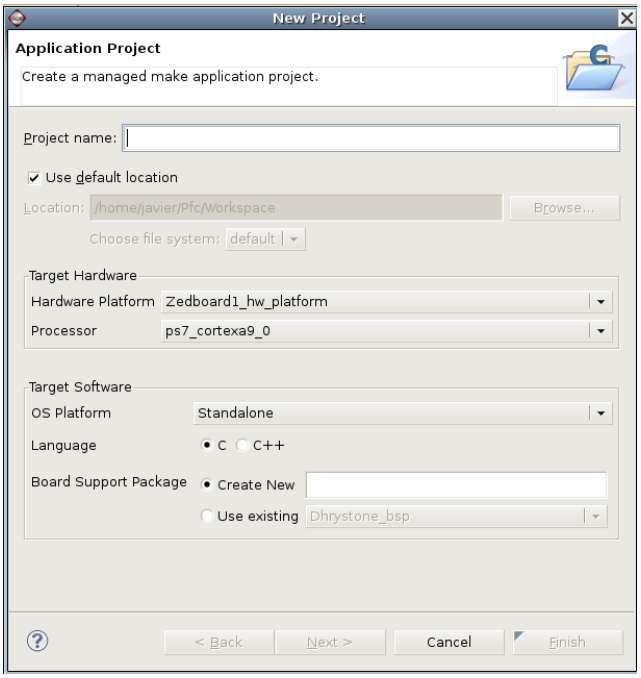


Ilustración 51: Creación nueva aplicación en XSDK

Para crear una aplicación será necesario **asociarla** a un **diseño hardware**, y a un procesador de ese diseño. Se selecciona un **lenguaje** de **programación** a elegir entre **C** o **C++**, y se indica si se quiere **autogenerar** un nuevo BSP o asignar uno creado anteriormente.

Finalmente se podrá elegir si se quiere usar algún tipo de plantilla para la aplicación. En nuestro caso se han utilizado **plantillas** para **agilizar** el proceso de prueba y así evitar el desarrollo de código propio. Una vez creada la aplicación, el siguiente paso es **configurar** la **compilación** del proyecto.

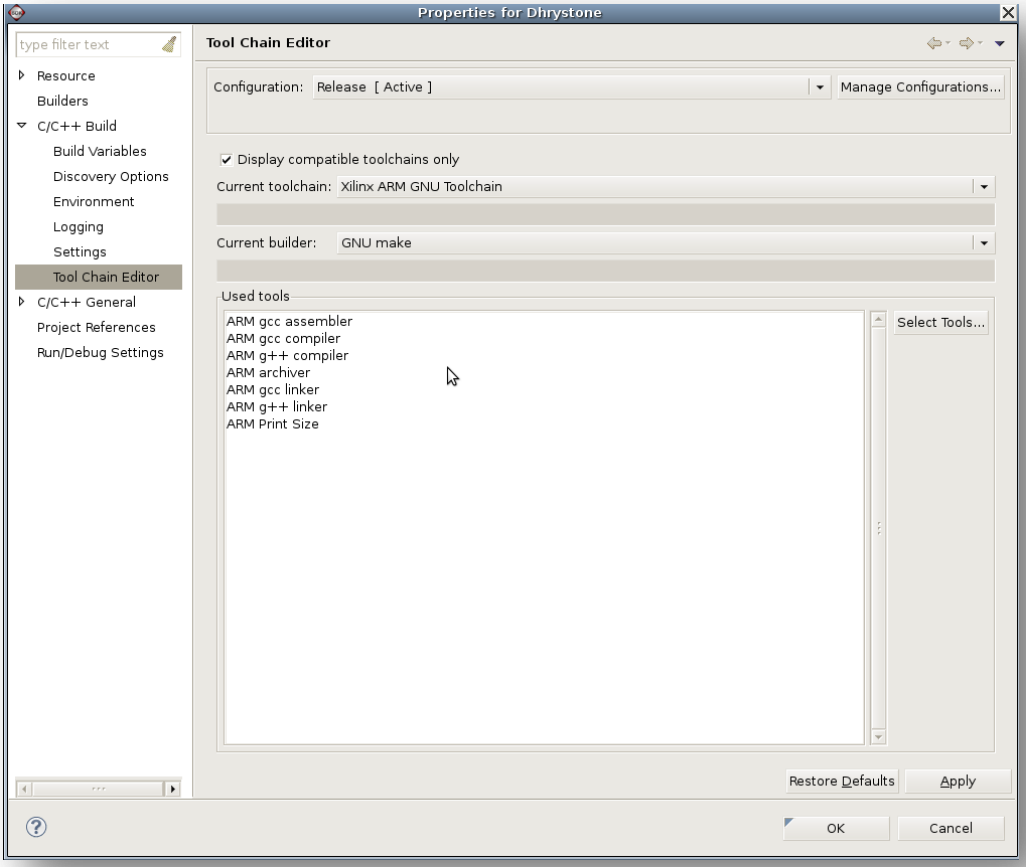


Ilustración 52: Configurar *toolchain* en XSDK

En nuestro caso es importante que la ***toolchain*** esté configurada para que utilice el **compilador GCC** para **ARM** en vez del compilador básico GCC. Para ello editaremos la opción de **propiedades** del menú ***Project***. Allí se seleccionará en el desplegable "**C/C++ Build**" y a continuación se comprobará que el *toolchain* sea el **Xilinx ARM toolchain** y que el constructor por defecto sea el ***make*** de **GNU**.

Una vez configurado el compilador, se intentará **construir** el **proyecto** sin fallos. A veces se puede producir un pequeño error en la construcción, dependiendo de la versión de GNU/Linux que haya instalada.

Este **error** viene debido a que algunas distribuciones GNU/LINUX nombran el comando para compilar con el **compilador GCC** como "make" y otras con "gmake". Debido a que XSDK usa "gmake", si la distribución utiliza el comando "make" se puede solucionar el error realizando un **enlace simbólico** del comando "gmake" al comando "make".

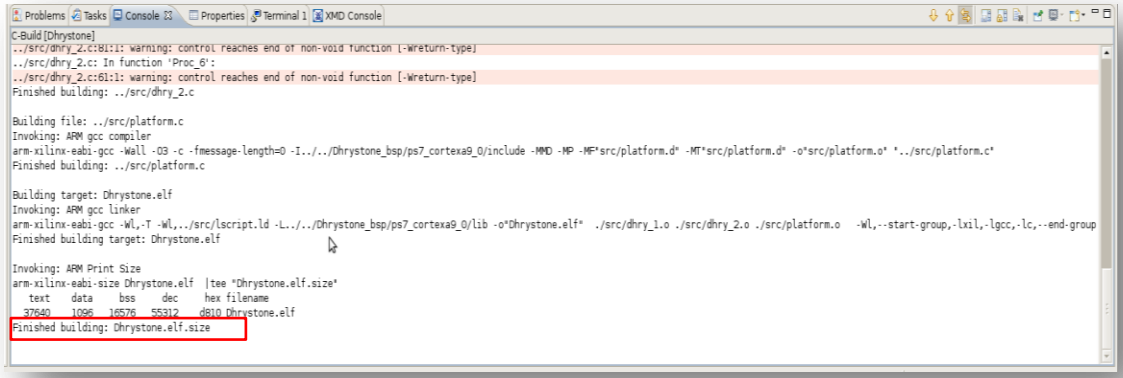


Ilustración 53: Resultado compilación correcta consola XSDK

Una vez **compilado** el código **correctamente**, se generarán **varios archivos** para poder **ejecutar** el **mismo** en el SoC de diversas maneras.

5.3 Implantación del sistema mediante JTAG

La **implantación** de la descripción hardware sistema en la placa de desarrollo Zedboard se va a realizar mediante la programación utilizando la interfaz **JTAG**. El proceso realizado será el de conectar el equipo de desarrollo desde un puerto **USB** a la placa de desarrollo a la interfaz JTAG de esta.

Una vez conectados ambos dispositivos se pasará a **descargar** el ***bitstream*** que define el sistema a la placa, que se encargará de programar **la definición del hardware** del sistema en la lógica programable del Zynq-7000. Para realizar la descarga de este fichero *bitstream* en la placa se puede realizar desde varias herramientas de la suite ISE, a pesar de eso, la herramienta diseñada para encargarse de la descarga del bitstream en la FPGA es la herramienta iMPACT.

5.3.1 Implantación en la placa mediante iMPACT

En primer lugar se realizará la **descarga** el diseño **hardware** utilizando **iMPACT**. La programación de la placa con el Impact requiere más pasos que en otros programas, pero se considera útil conocer su funcionamiento.

Al arrancar Impact se señalará que no queremos cargar un proyecto anterior. Posteriormente se intentarán **buscar** los **dispositivos** que hay conectados con el PC, utilizando el comando *Initialize Chain* que se puede ver dando un *click* derecho en la sección de ***Boundary Scan***.

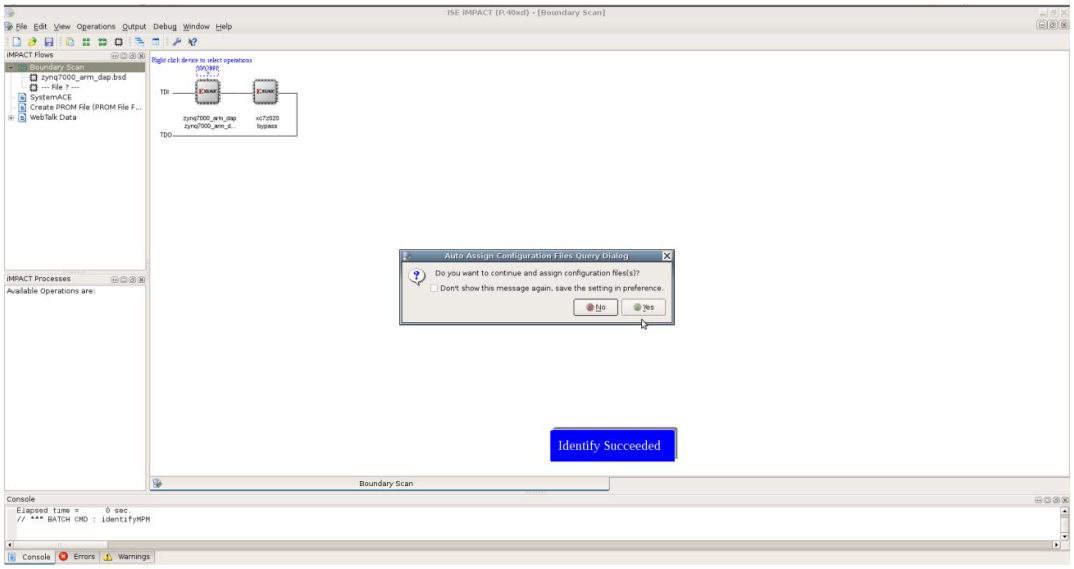


Ilustración 54: Reconocimiento de placa en Impact

Una vez reconozca la placa conectada al PC, mostrará en la pantalla principal los **dispositivos programables** de la placa y preguntará si se quiere continuar y asignar automáticamente la configuración a la misma, es decir, un diseño Hardware. En nuestro caso se dirá que no, para cargar el diseño de la placa de una forma manual.

Ya reconocidos los dispositivos programables en la placa se realizará un doble click en el dispositivo en el que se quiere programar.

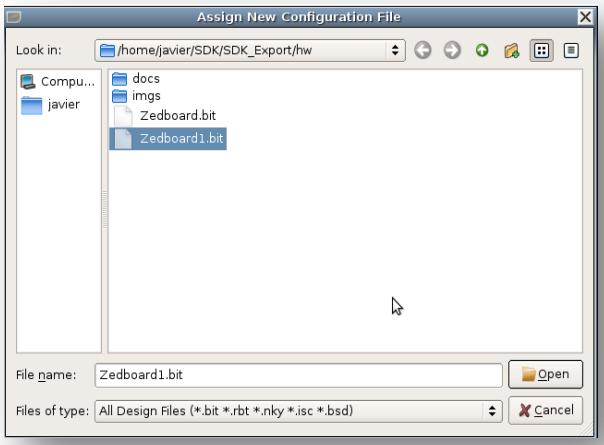


Ilustración 55: Selección de fichero de especificación de HW en Impact

Se introduce la ruta donde se encuentra el ***bitstream*** en el que se ha exportado el diseño hardware, normalmente situado en la ruta donde este almacenado el diseño hardware generado por el XPS.

Ya escogido el diseño hardware que se quiere programar en la placa, se pasará a realizar la programación. Para ello, se pulsará sobre el dispositivo que se va a programar, y se seleccionará el comando "Program".

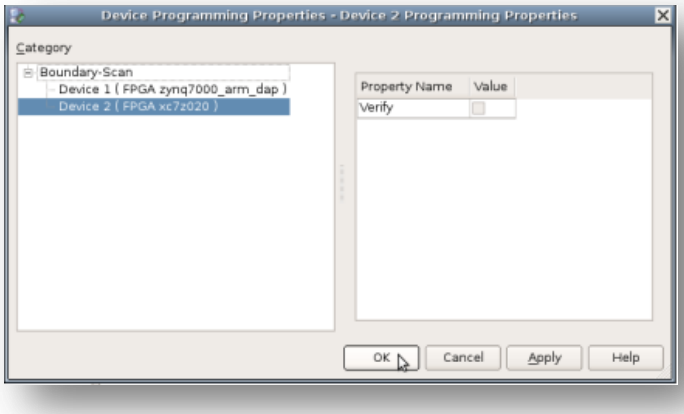


Ilustración 56: Programación de la placa en Impact

Finalmente, se mantendrán las opciones por defecto para la programación y se descargará el diseño hardware en la placa.

5.3.2 Implantación en la placa mediante XPS

A continuación se explicará cómo descargar el diseño hardware utilizando el XPS de una forma muy sencilla.

En primer lugar, una vez generado el fichero Bitstream, se comprobará que el fichero esta actualizado por si se ha realizado alguna modificación en el diseño HW. Para ello se usará el comando "Update Bitstream".

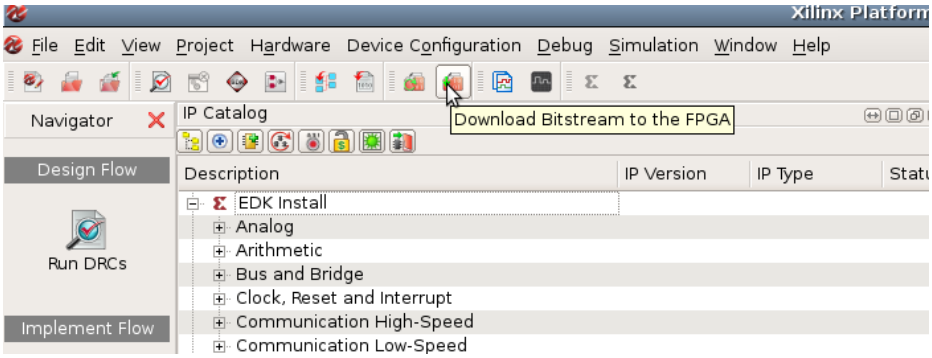


Ilustración 57: Descargar bitstream en FPGA mediante XPS

Con el fichero *bitstream* actualizado al diseño HW actual, se dará al comando "Download Bitstream", que descargará el diseño hardware en la placa, utilizando internamente el Impact.

5.3.3 Implantación en la placa mediante XSDK

La programación de la placa utilizando XSDK se realiza también de una forma sencilla e intuitiva.

Para programar el diseño HW en la placa, una vez se cuente con un diseño hardware en el espacio de trabajo del XSDK, se escogerá el comando "Program FPGA".

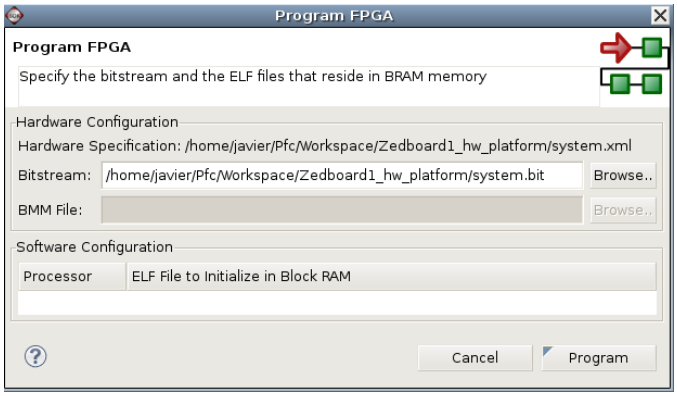


Ilustración 58: Programar FPGA mediante XSDK

Se selecciona el diseño HW en el que se va a programar, y en el caso de que te lo permita, se especificará el ejecutable ".elf". Si no se especifica ninguno o no deja introducir ninguno, se ejecutará el *bootloop*, y finalmente se programará la placa.

6 Pruebas

En este apartado se mostrará el **proceso** seguido para la creación de las **pruebas** unitarias de **simulación** sobre el periférico. Una vez explicado el proceso para la creación y ejecución de las pruebas, se mostrará el resultado de cada testbench y se explicarán los resultados obtenidos.

6.1 Implementación de pruebas unitarias

A continuación se explicará el proceso realizado para creación, implementación y ejecución de las **pruebas unitarias de simulación** sobre el periférico. Las pruebas se realizarán desde la herramienta ISE y su utilizad el simulador **iSIM**.

En primer lugar desde la pantalla principal de ISE, pasaremos a la pestaña de simulación, seleccionando en la opción encima de la jerarquía. Desde aquí le diremos que queremos **crear** un fichero Testbench en formato **VHDL**. Nombraremos el fichero y posteriormente seleccionaremos las entidades que deseamos tener en el testbench.

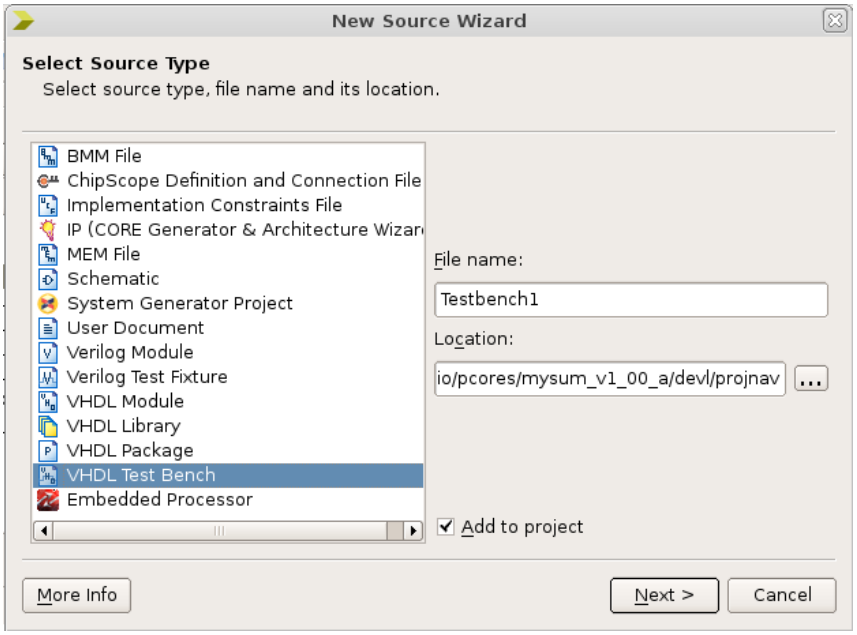


Ilustración 59: Creación testbench

Una vez creado el fichero, pasaremos a **implementar** alguno de los testbench diseñados. Para ellos podemos partir de la plantilla creada por el programa, o codificar la prueba de simulación desde cero.

Una vez terminado de codificar la prueba que se quiere realizar, comprobaremos que la sintaxis de la prueba es correcta, para ello realizaremos un chequeo de la sintaxis de la prueba. Una vez comprobada que la sintaxis es correcta, **ejecutaremos** la prueba en el **simulador** iSIM ejecutándola desde la pestaña de diseño del testbench.

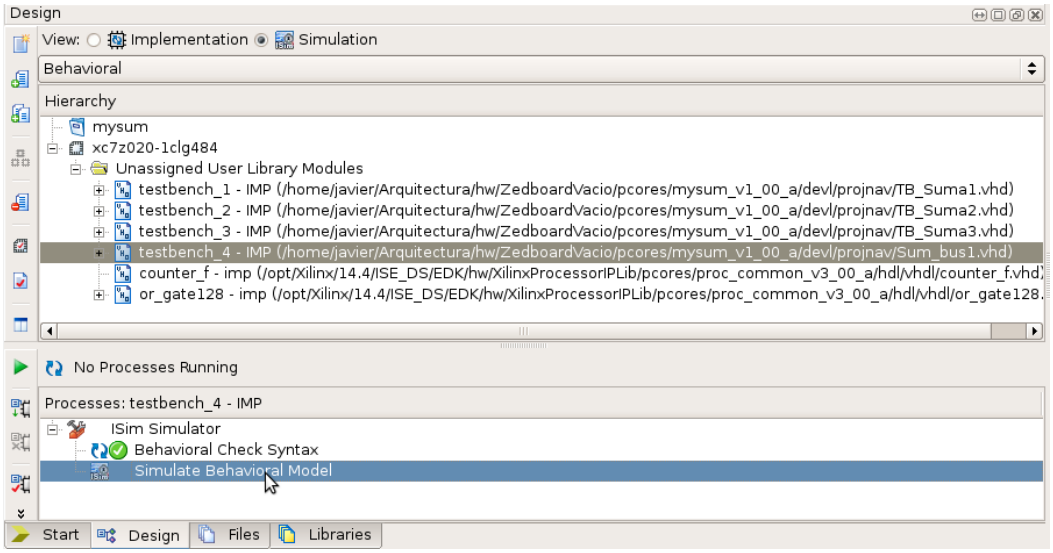


Ilustración 60: Ejecución de simulación de testbench

Una vez ejecutemos la prueba saldrá la pantalla principal del simulador. A la izquierda podremos encontrar las **entidades** a las que puede acceder el simulador, en la columna central todas las **señales** e **interfaces** de una **entidad** seleccionada y en la columna de la derecha el cronograma.

Para comprobar el resultado de una señal con el avance del testbench se puede arrastrar cualquier señal al cronograma para visualizar su resultado. Adicionalmente, permite cambiar el formato de los datos que se va a mostrar en el cronograma.

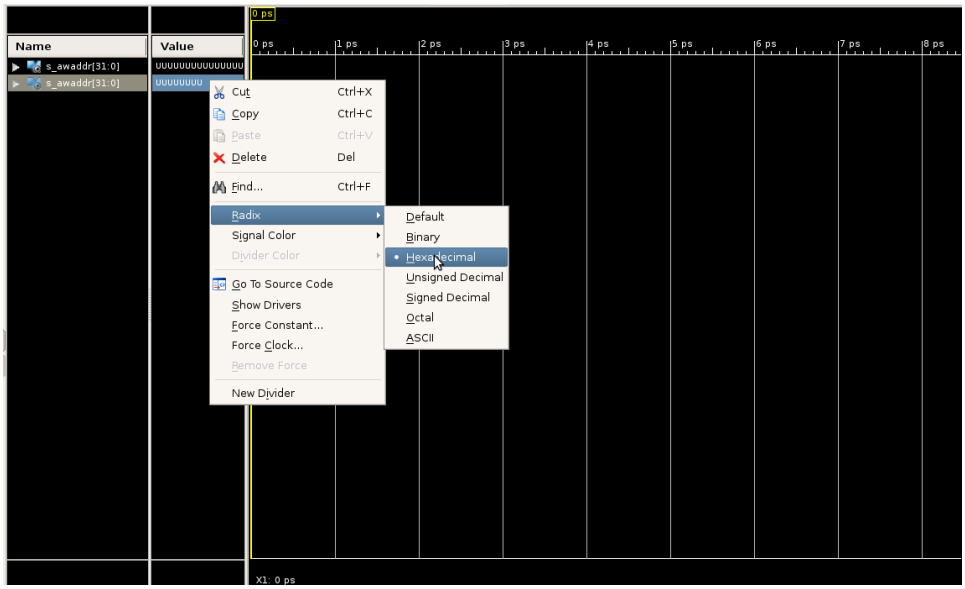


Ilustración 61: Cambiar el formato de los datos en iSIM

A continuación se mostrarán todos los cronogramas de salida correspondientes a cada uno de los testbench, y se razonarán los resultados obtenidos. Para obtener cronogramas de un tamaño reducido para facilitar su observación se han ejecutado los testbench mediante la **ejecución paso a paso.**

6.2 Testbench 1

Como se ha comentado en el apartado de diseño, este es el testbench más básico, simplemente escribe un dato en la lógica de usuario simulando una petición del módulo IPIF.

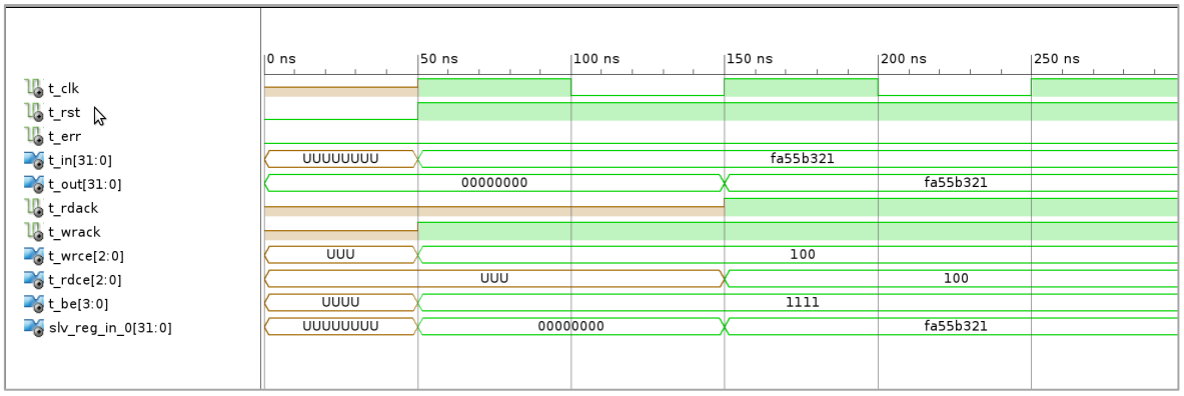


Ilustración 62: Cronograma Testbench 1

Como se puede observar en el diagrama, los valores de las señales de la lógica de usuario se inician con valores indeterminados 'U', para ello se utiliza la señal de reset. Cuando la señal de reset está a '0' y llega un flanco alto de reloj, se inicializan a 0 todos los registros de la lógica y adicionalmente se ponen los valores a su valor inicial.

Después de simular una señal de *reset* en la lógica de usuario, se indica a la lógica que se para de hacer el *reset*, que se puede observar porque el valor de la señal pasa de '0' a '1'.

El cometido principal del testbench es el de comprobar que se realiza correctamente una escritura. Como se puede observar en la señal t_wrce y t_rdce se ha elegido el primer registro tanto para lectura como escritura.

Se comprueba que el valor que se escribe en el registro "t_in", es el mismo que el valor que sale de la lógica al leer el registro "t_out".

6.3 Testbench 2

Este *testbench* tratará de comprobar que el funcionamiento del selector de bytes de una palabra que escribir, señal de "**byte enable**", funciona correctamente. Para ello se realizarán dos escrituras, una seleccionando todos los bytes de la palabra a escribir y en otra lectura escribiendo solo en los dos bytes más significativos.

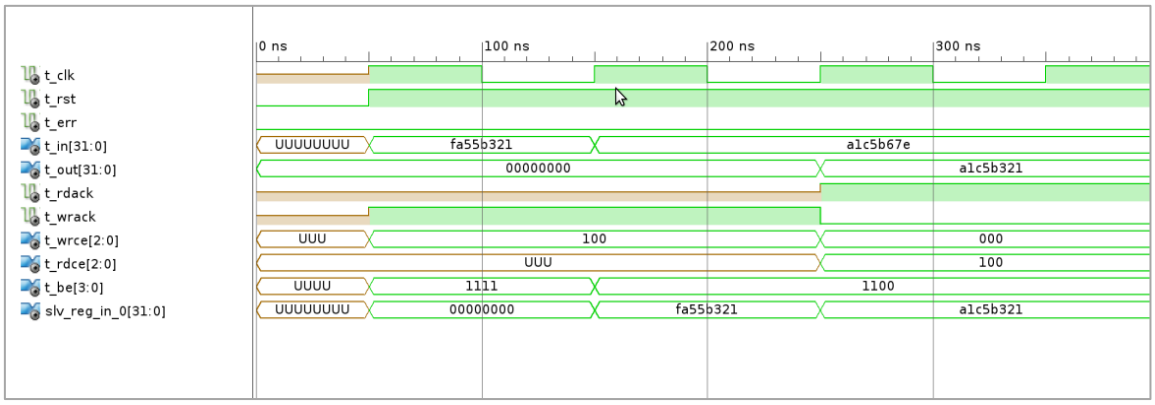


Ilustración 63: Cronograma Testbench 2

En el cronograma se puede apreciar la siguiente línea de ejecución:

1. En primer lugar se **resetea el dispositivo** poniendo la señal de reset a '0', y se desactiva la señal (valor '1') cuando llega el primer flanco de subida del reloj
2. **Introducimos el dato** con valor en hexadecimal x"FA55B321", en todos los bytes del registro seleccionado, el primer registro'100'. Se puede observar que es cuando se encuentra un flanco de subida de reloj cuando escribe el valor que recibe de la interfaz en el registro, cuando han transcurrido 150 ms
3. **Introduce otro dato** por interfaz para la lógica, pero en este caso solo decide escribir en los dos bytes más significativos de la palabra.
4. En el flanco de subida transcurridos 250 ns, **se pide una lectura del registro** 1, modificado dos veces. Se puede comprobar que los dos bytes más significativos pertenecen al dato más nuevo y los dos menos significativos al dato más antiguo.

6.4 Testbench 3

Este *testbench* trata de comprobar si realiza bien el procedimiento de la **suma** de los dos registros de la **lógica de usuario**. Para ello se simulará las **señales** recibidas por la lógica de usuario, proveniente del IPIF de **escritura y lectura**, y se comprobará que suma los dos registros sumandos correctamente en el resultado cuando no se realiza ninguna lectura. Para ello se **escribe** en los dos registros **sumandos**, se desactiva la lectura y posteriormente se realiza una **lectura** sobre el registro donde se almacena el **resultado**.

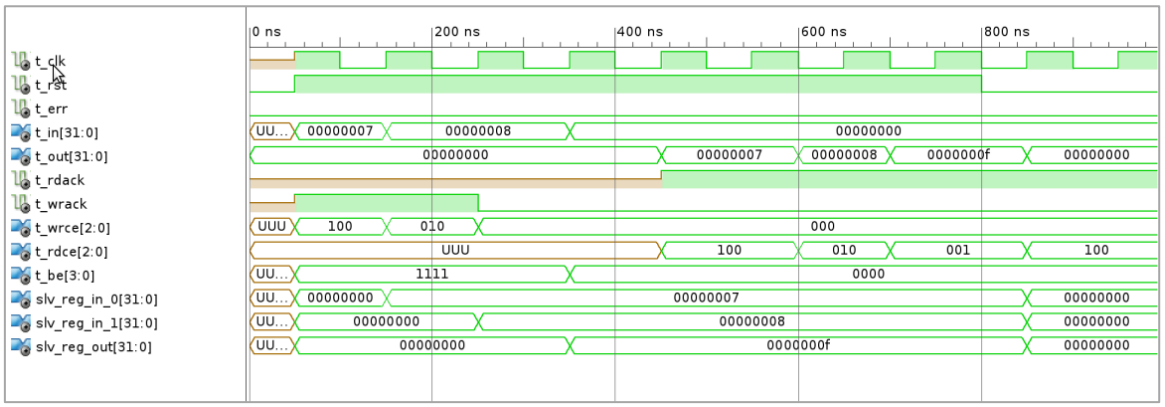


Ilustración 64: Cronograma Testbench 3

1. En primer lugar como en todos los *testbench* se envía una señal de reset para inicializar los registros.
2. Se desactiva la señal de reset y se escribe un dato enviado por la interfaz en el registro 1 en el siguiente flanco de subida
3. Se escribe otro dato enviado por la interfaz en el registro 2 en el siguiente flanco de subida
4. Se desactivan los selectores de registros debido a que no se van a hacer más escrituras. Esto provoca que en el siguiente flanco de subida se calcule la suma.
5. Se lee el dato del resultado de la suma, selecciona el 3 registro para leer y como se puede comprobar el dato de salida de la lógica (dato leído) es correspondiente a la suma de los datos anteriores.

Como se puede comprobar la suma se ha realizado correctamente ya que solo se suman los 4 bits menos significativos y en hexadecimal es fácil comprobar que el dato 00000007 + 00000008 = 0000000F, ya que 8+7 = 15 = F.

6.5 Testbench 4

En este *testbench* se comprobará el funcionamiento básico del periférico ante estímulos que provienen desde el bus del sistema. Este *testbench* se encargará de escribir un dato en un registro, y se comprobará que ese dato se escribe correctamente. En este caso se muestran todas las interfaces que provienen del bus del sistema, simulando información que pudiera venir desde el sistema

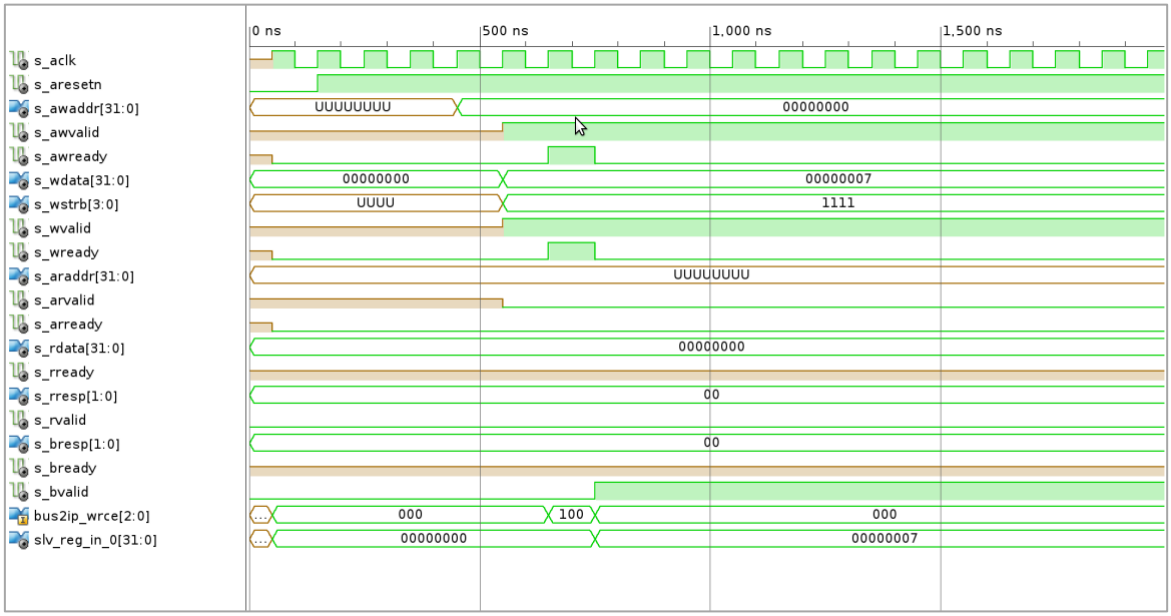


Ilustración 65: Cronograma Testbench 4

1. En primer lugar como en todos los casos se activa la señal de ***reset*** durante un ciclo para inicializar el periférico completo.
2. Posteriormente se simula una **petición** de **escritura** en una dirección de memoria x"00000000", esta dirección de memoria corresponde a una dirección de memoria del periférico y específicamente al primer registro.
3. Indicamos al periférico que la **dirección** de memoria se ha introducido correctamente en el bus y es **válida**. Para ello **activamos** la señal s_awvalid.
4. Debido a que vamos a realizar una escritura, y la lectura y escritura son excluyentes, indicamos que no se puede cargar una dirección de memoria de **lectura**. Para ello **desactivamos** la señal s_arvalid.
5. Mientras **esperamos** la señal de **confirmación** de la dirección de escritura, introducimos una **palabra** en la interfaz de datos de **escritura** del bus.
6. Indicamos que el **dato** que se quiere escribir se ha introducido correctamente en el bus y que el dato es **válido**. Para ello se activa la señal s_wvalid.
7. Cuando el periférico está **preparado** para escribir el dato, se encarga de escribir el dato en el registro correspondiente. Se puede comprobar que el valor del primer registro "slv_reg_in_0" contiene el dato introducido.

6.6 Testbench 5

En este *testbench* se quiere comprobar el correcto funcionamiento del selector de bytes de escritura, y comprobar el funcionamiento de lectura. Para ello se realizarán dos escrituras con diferentes valores en el selector de bytes, y se realiza una lectura posteriormente para comprobar el funcionamiento de la lectura.

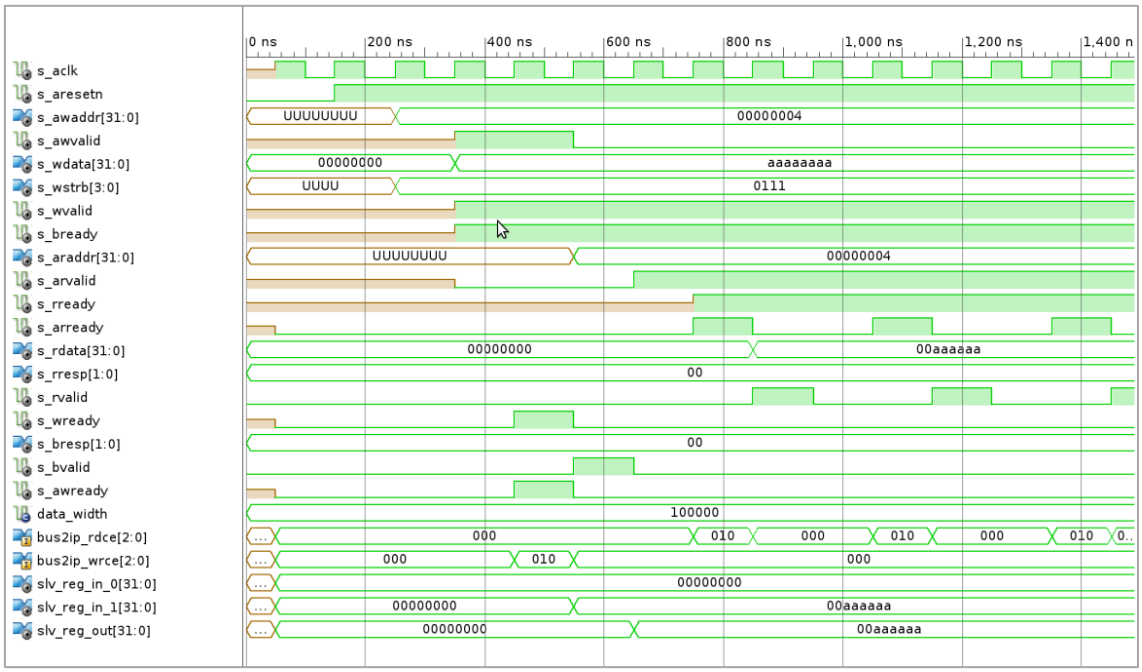


Ilustración 66: Cronograma Testbench 5

1. En primer lugar se realiza el **reset** como en todas las pruebas.
2. Posteriormente intenta escribir un dato en la **dirección** X“00000004”, como en el anterior caso **activa** la señal de “valid” de direcciones de **escritura** y **desactiva** la de **lectura**.
3. Se introduce un **dato** en el bus y se indica que es correcto y **válido**.
4. Se indica que **no** se va a introducir el **byte más significativo** de la palabra.
5. El dispositivo indica que está **preparado** porque el proceso se ha realizado correctamente y escribe el dato en el registro.
6. Se indica que se quiere realizar una **lectura** en la dirección de memoria en la que se ha escrito anteriormente, correspondiente al segundo registro.
7. Indicamos que no se va a escribir para poder realizar correctamente la lectura, se **desactiva** la señal “valid” de **escritura** y se **activa** la de **lectura**.
8. Se realiza la lectura en el mismo registro en el que se ha escrito para comprobar que el dato escrito es correcto. Para ello se introduce la dirección de memoria del registro en la interfaz de direcciones de lectura.
9. Se lee el valor del registro y se recibe por el bus de datos, el maestro habría leído correctamente el valor del registro.

6.7 Testbench 6

Este *testbench* va a ser el encargado de comprobar el correcto funcionamiento de la suma en el periférico. Para ello se va a simular la escritura de un dato en cada uno de los sumandos y posteriormente se simulará una lectura para leer el resultado de la suma, y comprobar que realiza y almacena la suma correctamente.

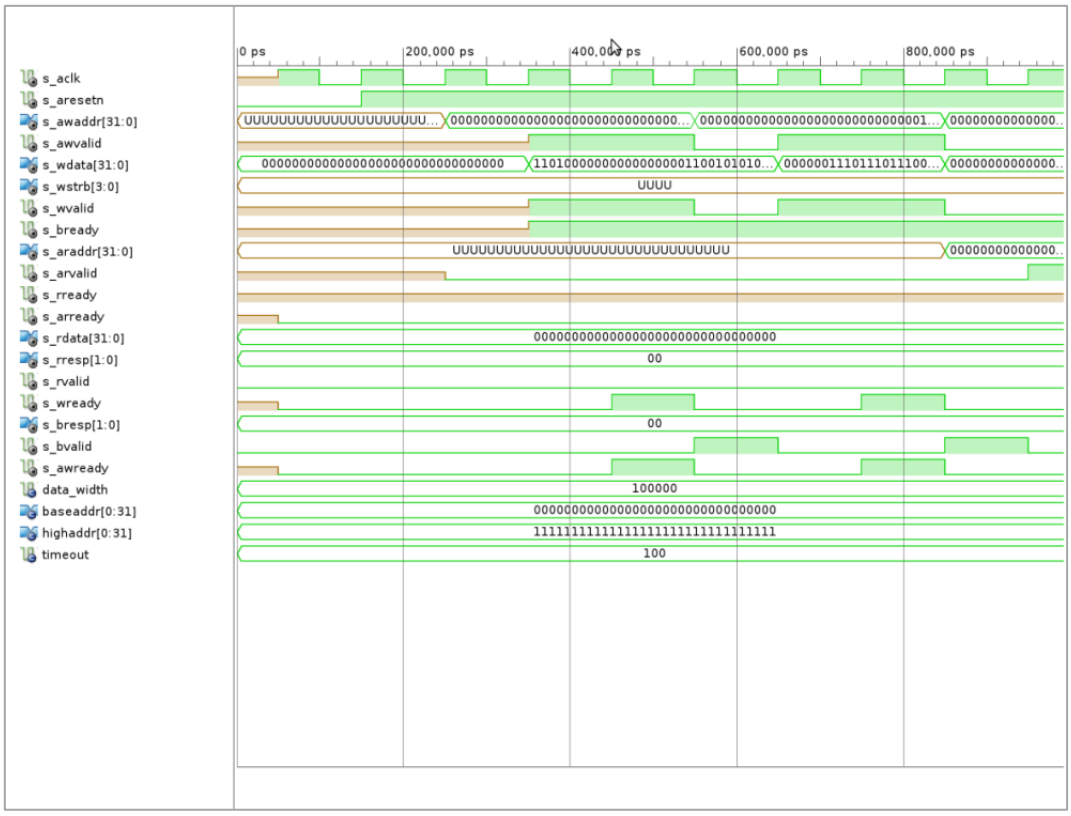


Ilustración 67: Cronograma Testbench 6

Mediante el proceso de lectura y escritura explicado anteriormente más detenidamente, se ha comprobado que se ha realizado la suma con éxito. Para ello se ha escrito los sumandos en el registro 0 -> dirección x"00000000" y el registro 1 -> x"00000004" y posteriormente se ha leído en el resultado, correspondiente al registro 2 del periférico, para ello se accede a la tercera dirección de memoria asignada, la dirección x"00000008".

Para facilitar ver que se ha realizado la suma correctamente, se han utilizado dos números sencillos para ver el **resultado** correctamente. Como se puede comprobar el dato leído en la dirección de memoria del registro 2 corresponde a la **suma** de los dos datos escritos en las direcciones de memoria donde se ha escrito anteriormente. Por lo tanto mediante este *testbench* se comprueba el correcto funcionamiento del proceso de suma.

6.8 Testbench 7

Este *testbench* realiza un **test** de **stress** sobre el dispositivo. En este *testbench* se sube la **frecuencia** del reloj de **10 MHz** a la velocidad a la que suele funcionar el reloj del sistema, **100 Mh**. Esta prueba se realiza para comprobar la **robustez** del periférico a una frecuencia más alta y comprobando su funcionamiento en la frecuencia ideal del periférico.

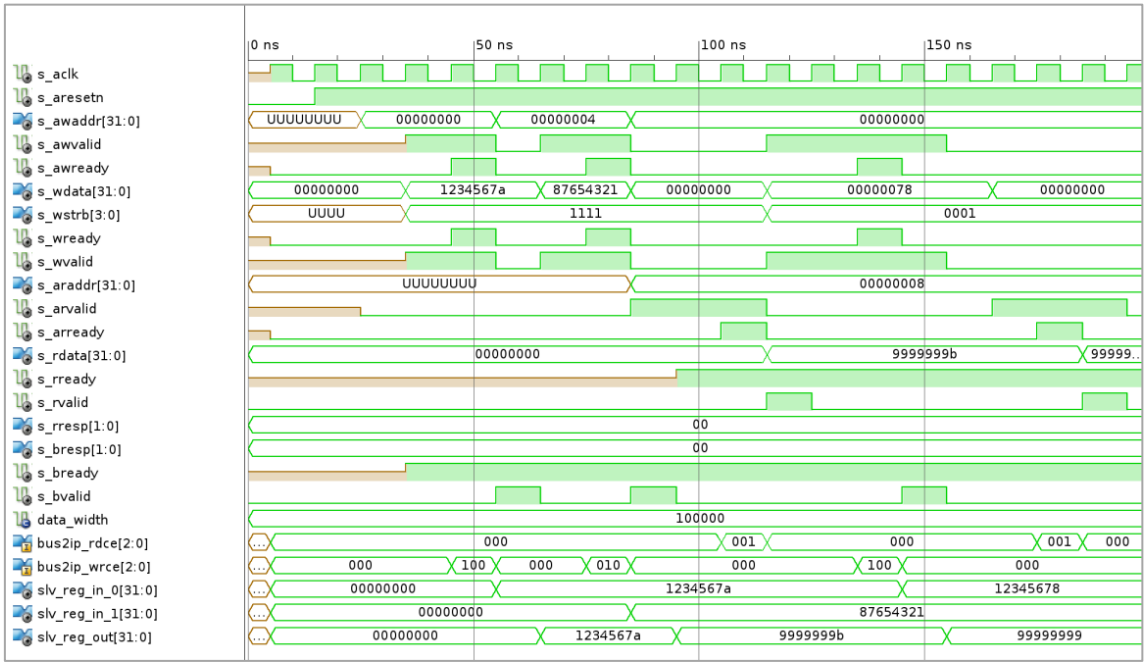


Ilustración 68: Cronograma Testbench 7

En el cronograma se puede apreciar respecto al resto fácilmente el **aumento** de la **frecuencia** del reloj. Para comprobar el funcionamiento del periférico ante una mayor velocidad de peticiones de escritura y lectura en menos tiempo, se han simulador varias peticiones de escritura y lectura sobre direcciones de memoria asignadas al periférico.

En el cronograma se puede observar que se han realizado en primer lugar dos escrituras de todos los bytes en los registros 0 y 1. Después se ha leído el registro 3 comprobando el valor de la suma realizada, como se puede apreciar el valor de la suma se calcula y lee correctamente.

Posteriormente se modificará el byte menos significativo del registro 1, realizando una escritura sobre esa dirección de memoria. Finalmente se realizará otra lectura sobre el registro 3, comprobando que el valor de la suma se ha modificado correctamente.

7 Planificación y presupuesto

Es este aparatado se explicará la planificación detallada de las tareas de **investigación**, **desarrollo** y **documentación** del proyecto. Además se explicarán los **costes** estimados del proyecto, teniendo en cuenta todos los costes necesarios para su realización.

7.1 Planificación

En este apartado se explicará la **planificación** realizada y seguida para la realización del proyecto. Para explicar la planificación del proyecto se detallarán todas las **tareas** y **fases** por las que está compuesta la realización del proyecto, y posteriormente se plasmarán su **duración** y **relación** mediante un diagrama de Gantt

La duración del proyecto ha sido aproximadamente de 8 meses, habiendo comenzado el proyecto a principios de noviembre de 2012, y finalizado a finales de Junio de 2013.

Las fases de desarrollo del proyecto y en particular las tareas por las que está compuesto son las siguientes:

- **Propuesta del proyecto:** esta tarea consiste principalmente en la elección y selección del trabajo que se quiere realizar sobre la placa Zynq y definiendo el proyecto que se quiere realizar.
- **Puesta a punto del sistema:** En esta tarea se realizará la instalación de todas las herramientas software necesarias, en primer lugar para la toma de contacto con las FPGAs, y después la instalación del entorno de desarrollo escogido.
- **Análisis:** en esta tarea se realizará todo el apartado con el estudio, investigación y especificación de las características, objetivos y requisitos del proyecto. Esta tarea se puede dividir en varias subtareas:
    - **Toma de contacto con los dispositivos programables:** esta tarea consiste en la toma de contacto con toda la tecnología de los dispositivos programables realizando un análisis e investigación sobre todos sus aspectos. Leyendo documentación oficial de sus especificaciones, analizando y probando las diferentes herramientas de desarrollo y probando el desarrollo en estos dispositivos.
    - **Análisis e investigación del proyecto:** en esta tarea se realizará una investigación y análisis más específico sobre el sistema que se quiere realizar, mientras que va avanzando la definición de la propuesta y los objetivos del proyecto.

  - **Elección del entorno operacional**: en esta subtarea se analizará y escogerá el entorno operacional a utilizar, estudiando todas sus alternativas.
  - **Definición capacidades y restricciones**: en esta tarea se definirán todas las capacidades y restricciones que queremos que cumpla con el desarrollo del proyecto.

- **Diseño:** durante el desarrollo de esta tarea se ha diseñado y definido el funcionamiento y la estructura del sistema a desarrollar, así como de las pruebas que se quieren realizar. La tarea del diseño se subdivide en varias tareas:
  - **Diseño del sistema a implementar.**
  - **Diseño del periférico.**
  - **Diseño de las pruebas unitarias.**

- **Implementación:** durante esta fase se realizará la implementación de todo el diseño del sistema, en esta fase también se investigará y buscará información sobre cómo implementar el diseño creado mediante documentación oficial o tutoriales. Esta tarea se puede dividir en varias tareas:
  - **Aprendizaje lenguajes de programación:** durante esta tarea se ha aprendido el funcionamiento del lenguaje de descripción de hardware escogido VHDL.
  - **Implementación periférico:** en el desarrollo de esta tarea comprende la creación, implementación y síntesis del periférico desarrollado.
  - **Implementación sistema:** en el transcurso de esta tarea engloba la creación el sistema completo, la importación del periférico, la configuración del sistema y la exportación de este.
  - **Implantación del sistema:** esta tarea consiste en implantar el sistema realizado en la placa de desarrollo.

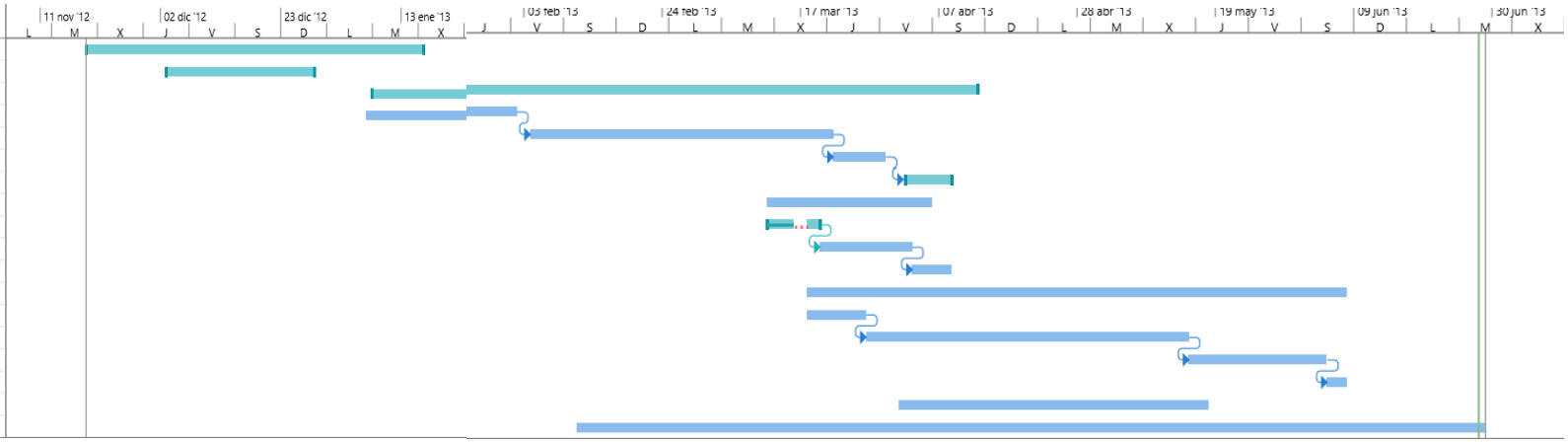
- **Prueba:** esta tarea engloba la creación, implementación y ejecución de las pruebas unitarias del simulador.
- **Documentación:** Esta tarea representa la documentación de todo el proceso realizado para la realización del proyecto.

La escala del tiempo realizado de todas las tareas es el siguiente:

Ilustración 69: Escala de tiempo del proyecto

El **diagrama de Gantt** que describe la planificación del proyecto se puede observar en la siguiente página.

Id	Nombre de tarea	Comienzo	Fin
1	Propuesta del proyecto	lun 19/11/12	mié 16/01/13
2	Puesta a punto Sistema	lun 03/12/12	vie 28/12/12
3	Análisis	mar 08/01/13	vie 12/04/13
4	Toma de contacto dispositivos programables	lun 07/01/13	vie 01/02/13
5	Análisis e investigación del proyecto	lun 04/02/13	jue 21/03/13
6	Elección entorno operacional	vie 22/03/13	vie 29/03/13
7	Definición capacidades y restricciones	mar 02/04/13	lun 08/04/13
8	Diseño	mar 12/03/13	vie 05/04/13
9	Diseño del sistema a implementar	mar 12/03/13	mar 19/03/13
10	Diseño del periférico	mié 20/03/13	mar 02/04/13
11	Diseño de las pruebas unitarias	mié 03/04/13	lun 08/04/13
12	Implementación	lun 18/03/13	vie 07/06/13
13	Aprendizaje lenguajes de programación	lun 18/03/13	mar 26/03/13
14	Implementación Perirérico	mié 27/03/13	mar 14/05/13
15	Implementación sistema	mié 15/05/13	mar 04/06/13
16	Implantación del sistema	mié 05/06/13	vie 07/06/13
17	Prueba	lun 01/04/13	vie 17/05/13
18	Documentación	lun 11/02/13	vie 28/06/13



7.2 Presupuesto

Es este aparatado se especificarán una estimación de los **costes** generados con la realización de la totalidad del proyecto. Los costes del proyecto se pueden agrupar en tres tipos de costes: **costes de personal**, **costes de hardware** y **costes de software**.

7.2.1 Costes de personal

El desarrollo del proyecto ha sido cubierto por **una sola persona** encargada de realizar todo el trabajo durante toda la duración del proyecto. A pesar de esto, debido a que esta persona ha tenido que desempeñar **diferentes roles** a la hora del desarrollo del proyecto se han definido distintos tipos para la realización del presupuesto.

Los tres distintos roles que se han tenido que cubrir para la realización del proyecto son las siguientes:

- **Técnico informático**: Técnico encargado de la **puesta a punto** del entorno de desarrollo del sistema, para asegurar su correcto funcionamiento.
- **Ingeniero hardware**: Ingeniero encargado del **diseño** y sobre todo **implementación** del **hardware** del sistema a bajo nivel, específicamente del periférico realizado.
- **Ingeniero informático sénior**: este rol se ha encargado de realizar la mayor cantidad de tareas del proyecto, que engloba el proceso de **investigación**, **diseño**, **implementación** y **análisis** del sistema, **pruebas** y **documentación**.

Concepto	Salario/hora	Número de horas	Total
Ingeniero sénior	44,00€	440 h	19.360,00€
Ingeniero hardware	50,00€	250 h	12.500,00€
Técnico informático	24,00€	70 h	1.680,00€
			Total: 33.540,00€

7.2.2 Costes de hardware

Este tipo de coste engloba todos los gastos realizados en la adquisición del material **hardware** para montar el **entorno operacional**. Todos los precios incluidos en la tabla tienen incluido el IVA correspondiente.

Concepto	Precio
Ordenador portatil Sony Vaio SE 15''	1.200€
Placa de desarrollo Digilent Zedboard	320 €
	Total: 1.520,00€

7.2.3 Costes de software

En este tipo de coste vamos a calcular el **precio** de las **licencias** del **software** necesario para el desarrollo del proyecto realizado. La mayor parte del proyecto se ha realizado con **licencias** totalmente **gratuitas** y **libres**, y el resto de programas se han utilizado **licencias académicas** que no han tenido coste ninguno. A pesar de eso, se ha calculado el coste de estas licencias en el caso de que se quiera diseñar el proyecto para un uso no académico.

Concepto	Precio
Xilinx ISE Design Suite 14.3	2.861,41€
Microsoft Office Professional 2010	699,90€
	Total: 3.561,31€

7.2.4 Presupuesto total

Una vez estimados todos los costes del proyecto, se calcula la **suma** de los totales de sus **costes**. A la suma de los costes de todos los tipos se le añadirá un porcentaje para cubrir los **riesgos** que puedan surgir durante la realización del proyecto, y otro porcentaje para asumir los **beneficios** que se quieren obtener gracias a la realización del proyecto. El presupuesto final es el mostrado a continuación:

Concepto	Total
Costes de personal	33.540,00€
Costes de hardware	1.520,00€
Costes de software	3.561,31€
Subtotal	38.621,31€
Riesgo (10%)	3.862.13€
Beneficio (20%)	7.724,26€
	Total: 50.207.07€

El presupuesto total del desarrollo de este proyecto asciende a la cantidad de **CINCUENTA MIL DOSCIENTOS SIETE COMA SIETE euros incluyendo costes indirectos**.

8 Conclusiones

Con la realización de este trabajo dirigido se ha **alcanzado los objetivos** que se habían planteado, entre ellos la **toma de contacto** con la placa **Zedboard** y algunas de sus **herramientas** de desarrollo, así como el desarrollo del módulo para el Zynq.

Se han cumplido los principales objetivos que se querían alcanzar y marcados al inicio, entre ellos la **investigación sobre el comportamiento** a todos los niveles del **Zynq-7000**. La implementación del sistema y el módulo hardware, como se comentó en los objetivos, han cumplido el objetivo de ser útiles como un **punto de partida** para poder **desarrollar un producto en el futuro** para un cliente.

La idea original era realizar alguno de los trabajos futuros adicionalmente como objetivos iniciales, pero debido a excesos en la **extensión** de **trabajo** y falta de **tiempo** una vez realizada la planificación, se ha decidido terminar el proyecto en este punto y no extenderse en uno de los campos. Esto se ha decidido debido a que se ha querido adaptar la cantidad de trabajo y de tiempo a la cantidad óptima y proporcional con un proyecto de final de carrera.

8.1 Trabajos Futuros

Como se comentó en la introducción, la principal **motivación** para la realización de este proyecto es la de poder saber aprovechar e investigar el funcionamiento y las posibilidades del Zynq. Por lo que la mayoría de los trabajos futuros que se podría realizar a partir de este proyecto son una gran cantidad.

El trabajo futuro más lógico inmediatamente después de la realización del proyecto de investigación, sería utilizar estos conocimientos y la información sintetizada con el proyecto para la implementación de una solución para un cliente.

En el caso de que no se tenga ningún proyecto de ningún cliente, se podría seguir explotando el potencial del dispositivo. Las **principales ideas** para un trabajo futuro, que se pudieran continuar como un proyecto de investigación son las siguientes:

- **Carga de Sistemas Operativos**.
- **Creación de driver en SSOO del IP.**
- **Realización de código empotrado uilizando el IP.**
- **Creación de otros tipos de IP.**
- **Comparativa con otros SoC**.
- **Utilización de otras herramientas de desarrollo**.
- **Utilización de la Suite de diseño Vivado HLS**.

A continuación se explicarán más detalladamente algunos de los trabajos futuros enunciados.

Creación de software standalone - empotrado

El trabajo futuro más inmediato a la hora de realizar este proyecto de investigación, es el del desarrollo de un **módulo hardware** y un **sistema** del Zynq-7000 para adaptarse a las exigencias de un cliente.

Utilizando como base este proyecto de investigación, la idea sería intentar dar una solución para un problema propuesto por un cliente. El desarrollo de algún IP pedido por algún cliente, o el diseño e implementación de un sistema para el Zynq-7000.

Carga y utilización bajo sistemas operativos

El dispositivo Zynq-7000 tiene suficiente potencia para poder cargar una gran variedad de sistemas operativos, tanto sistemas operativos empotrados, como sistemas operativos móviles (***Android***), y sistemas reducidos de sistemas operativos convencionales (***Linux***, ***Windows***). Tener un sistema operativo instalado, permitiría una mayor variedad y **facilidad** a la hora del **desarrollo software** para aprovechar el hardware diseñado para el sistema.

Debido a eso gracias al **compilador cruzado** provisto por el fabricante del procesador del Zynq-7000(ARM), se puede compilar un sistema operativo compilado para el diseño hardware diseñado. Así mismo gracias al compilador, permite compilar código para el dispositivo con el sistema operativo implementado.

Debido a que el dispositivo tiene interfaces para un lector de tarjetas Flash, permite cargar sistemas más pesados, almacenando el SSOO en una **tarjeta SD**. Esto se puede realizar introduciendo un pequeño cargador de SSOO ejecutado directamente mediante el JTAG o almacenado en la memoria Flash, capaz de poder cargar un sistema más pesado almacenado en una tarjeta SD

Hay que remarcar, que la placa Zedboard elegida para probar el desarrollo, contiene todas las interfaces y potencia para poder cargar diferentes SSOO.

Creación de drivers - daemons para periféricos IP

Una de las posibilidades de un trabajo futuro sería el desarrollo de un ***driver*** o ***daemon*** para uno de los sistemas operativos enunciados anteriormente. El desarrollo de un driver permitiría que el código software pudiera **acceder** al **periférico** a través del **sistema operativo**.

Esto sería una grandísima herramienta para agilizar el desarrollo de software para sistemas que contengan un IP desarrollado por nosotros. Esto permitiría que se pudiera realizar distintos tipos de software que sean capaces de interactuar con el periférico.

Uso de diferentes herramientas y uso de lenguajes HLS

El SoC Zynq como se comentó en el correspondiente apartado en el estado del arte del proyecto, contiene múltiples herramientas capaces de aprovechar su dualidad de **desarrollo hardware/software.**

Se puede investigar la utilización de otras herramientas para el desarrollo de software y hardware, realizando un proyecto parecido de un módulo y un sistema similar. Entre unas de las herramientas, resulta altamente interesante la utilización de las herramientas de desarrollo con **lenguajes de alto nivel de síntesis**, como la herramienta de Xilinx Vivado HLS.

Una buena opción de trabajo futuro, sería partiendo del trabajo de investigación realizado, realizar una comparativa y *benchmarking* entre las diferentes herramientas de desarrollo para el Zynq, comparando sus funcionalidades, tiempos de generación y otras medidas algo más subjetivas como la facilidad de configuración o desarrollo.

Una buena opción de trabajo futuro, sería partiendo del trabajo de investigación realizado, realizar una comparativa y *benchmarking* entre las diferentes herramientas de desarrollo para el Zynq, comparando sus funcionalidades, tiempos de generación y otras medidas algo más subjetivas como la facilidad de configuración o desarrollo.

Comparativa con otros dispositivos

Una buena idea sería la comparativa entre otras plataformas y SoC de **consumo** y **tamaño** similar, dispositivos de consumo medio-bajo. Para ello se podrían comparar las características técnicas de los diferentes dispositivos o el desarrollo de benchmarks.

Mediante *benchmarks* se pueden comparar el rendimiento de estos dispositivos de corte similar. Para la realización de estos benchmarks sería realmente útil la implementación de memoria extra o aceleradores y comprobar su efectividad.

Utilización de partial recofiguration

Los nuevos dispositivos de FPGA y dispositivos programables permiten la utilización de una tecnología llamada "*partial reconfiguration"*. Esta tecnología permite la reprogramación de una lógica programable de una forma parcial, reconfigurando sólo una parte de la FPGA dejando el resto de la lógica intacta.

Esta tecnología provee muchas posibilidades, una de las más interesantes es la de poder **añadir bloques completos** o periféricos como IPs mientras la FPGA de un SoC **está en funcionamiento**.

Como un proyecto de investigación muy prometedor sería la implementación de un sistema capaz de añadir periféricos en caliente utilizando la "*partial reconfiguration*" ,durante la ejecución de un sistema operativo, generando y compilando automáticamente los drivers asociados a este periférico. Para ello sería unificar todos varios trabajos futuros comentados anteriormente para comprobar su viabilidad al ser un proyecto de gran tamaño y de alta extensión, para comprobar el rendimiento antes de realizar el proyecto.

Además se podría adaptar esta idea a las necesidades de un proyecto software que necesite cambios de hardware y diferentes módulos durante su ejecución para optimizar en mayor medida su rendimiento.

9 Acrónimos, abreviaturas y definiciones

Acrónimos y abreviaturas

- **AMBA**: Advanced Microcontroller Bus Architecture
- **AMD**: Advanced Micro Devices
- **ARM**: Advanced RISC Machine
- **ASIC**: Application Specific Integrated Circuit
- **AXI**: Advanced eXtensible Interface
- **BSP**: Board Support Package
- **CISC**: Complex Instruction Set Computing
- **CODEC**: COder-DECoder
- **CPLD**: Comples Programmable Logic Device
- **DDR**: Double Data Rate
- **DSP:** Digital Signal Processing
- **EEPROM**: Electrically Erasable Programmable ROM
- **ELF**: Executable and Linking Format
- **ESA**: European Space Agency
- **FPGA**: Field-Programmable Gate Array
- **FPU**: Floating-Point Unit
- **GAL**: Generic Array Logic
- **GCC**: GNU Compiler Collection
- **GPU**: Graphics Processing Unit
- **HDL**: Hardware Description Language
- **HDMI**: High-Definition Multimedia Interface
- **HLL:** High Level programming Lenguaje
- **HLS:** High Level Synthesis
- **IO**: Input/Output
- **SSOO**: Sistemas Operativos
- **IDE**: Integrated Development Environment
- **ISE**: Integrated Software Environment
- **IP** device: Intellectual Property device
- **JTAG**: Joint Test Action Group
- **KB**: KiloByte
- **LED**: Light Emitting Diode
- **MB**: MegaByte
- **Mb**: MegaBit
- **MHz**: MegaHertzio
- **MIO**: Multiplexed Input Output
- **OLED**: Organic Light Emitting Diode
- **PAO**: Peripheral Analysis Order file
- **PAL**: Programmable Array Logic
- **PC**: Personal Computer
- **PL**: Programmable Logic

- **PLD**: Programmable Logic Device
- **PLL**: Phase Locked Loop
- **PMod**: Peripheral Module
- **POSIX**: Portable Operating System Interface
- **PS**: Processor System
- **QSPI**: Queued Serial Peripheral Interface
- **RAM**: Random Access Memory
- **RISC**: Reduced Instruction Set Computing
- **ROM**: Read-Only Memory
- **SDK**: Software Development Kit
- **SoC**: System on Chip
- **UART**: Universal Asynchronous Receiver/Transmitter
- **UART2USB**: UART to USB interface
- **USB**: Universal Serial Bus
- **USB OTG**: USB On The Go
- **VGA**: Video Graphics Array
- **VHDL**: VHSIC Hardware Description Languaje
- **VHSIC:** Very High Speed Integrated Circuit
- **WiFi**: Wireless Fidelity
- **XPS**: Xilinx Platform Studio
- **XSDK**: Xilinx Software Development Kit

Definiciones

- **Active-Low:** Señal de control que se activa cuando se recibe una señal con valor '0'.
- **Active-High:** Señal de control que se activa cuando se recibe una señal con valor '1'.
- **Android:** Sistema operativo de dispositivos móviles desarrollado por Google.
- **Baremetal:** Ejecución directa sobre el hardware de un dispositivo sin ningún sistema operativo o gestor por debajo.
- **Bus:** Medio de comunicación o de interconexión en electrónica, mediante un bus se pueden pasar varias señales simultáneamente.
- **Enlace simbólico**
- **Linux:** Sistema operativo de código abierto y software libre basado en **Unix.**
- **Sistema empotrado/embebido:** Se trata de un sistema de computación diseñado para desempeñar una funcionalidad muy específica, comúnmente utilizada en sistemas de tiempo real.
- **Soft Processors:** Procesador implementado en una lógica programable.
- **toolchain:** Grupo de herramientas que trabajan en cadena para conseguir la compilación de un código.
- **testbench:** Simulación de prueba de un IP.
- **Unix:** Sistema operativo de PC desarrollado por Bell Labs.
- **Windows: Sistema operativo de PC desarrollado por Microsoft.**

10 Bibliografía

[1] Enciclopedia sobre Field Programmable Gate Array (Español). 2013. Disponible: *http://es.wikipedia.org/wiki/FPGA*

[2] Enciclopedia sobre Field Programmable Gate Array (Inglés). 2013. Disponible: *http://en.wikipedia.org/wiki/FPGA*

[3] Breve historia de los Field Programmable Gate Array. 2012. Disponible: http:/www.semiwiki.com/forum/content/1596-brief-history-field-programmable-devices-fpgas.html

[4] Breve historia de los ASIC. 2012. Disponible: http:/www.semiwiki.com/forum/content/1587-brief-history-asic-part-i.html

[5] Productos y especificaciones de FPGAs de Xilinx. 2013. Disponible: *http://www.xilinx.com/products/silicon-devices/fpga/index.htm*

[6] Descarga de herramientas de diseño de Xilinx. 2013. Disponible: http://www.xilinx.com/support/download/index.htm

[7] Enciclopedia sobre Xilinx (Inglés). 2013. Disponible: *http://en.wikipedia.org/wiki/Xilinx*

[8] Enciclopedia sobre x86 (Inglés). 2013. Disponible: *http://en.wikipedia.org/wiki/x86*

[9] Enciclopedia sobre arquitectura MIPS (Inglés). 2012. Disponible: *http://en.wikipedia.org/wiki/MIPS_architecture*

[10] Enciclopedia sobre ARM (Inglés). 2013. Disponible: *http://en.wikipedia.org/wiki/ARM*

[11] Enciclopedia sobre PLD (Inglés). 2013. Disponible: *http://en.wikipedia.org/wiki/Programmable_Logic_Device*

[12] Procesadores y especificaciones de ARM. 2013. Disponible: *http://www.arm.com/products/processors/index.php*

[13] Enciclopedia sobre Soft Processors 2013. Disponible: *http://www.arm.com/products/processors/index.php*

[14] Especificación AMBA AXI4LITE desde la página ARM. Necesario registro en la página de ARM para su acceso: *https://silver.arm.com/download/download.tm?pv=1074010*

[15] VHDL: Programming by example. Perry, Douglas L. McGraw-Hill. ISBN: 0071400702

[16] Embedded core design with FPGAs. Navabi, Zainanlabedin. McGraw-Hill. ISBN: 9780071474818

[17] Digital Systems Design with FPGAs and CPLDs. Ian Grout. Elsevier. ISBN: 9780750683975

[18] Designing SOCS with configured cores: Unleashing the Tensilica Xtensa and Diamond Cores. Leibson, Steve. Morgan Kaufmann. ISBN; 0123724988:

[19] From Asics to Socs: A Practical Approach. Nekoogar, Fazard. Prentice Hall PTR. ISBN: 0130338575

11 Anexos

A continuación se proceden a detallar los anexos del proyecto. Dicha documentación que contiene tareas realizadas para la consecución del proyecto no fue considerado conveniente adjuntarlo en el proyecto pero se considera importante su consideración, aunque sea como un anexo al mismo.

11.1 Puesta a punto del entorno de desarrollo

Para la **instalación** del entorno operacional analizado en el punto 3.2 se ha necesitado realizar una puesta a punto del sistema. Se intentarán obviar los procedimientos no relevantes para la puesta a punto, como la instalación del sistema operativo, de un navegador o muchas instalaciones necesarias para poner a punto cualquier PC. Esta instalación se va a realizar sobre un SSOO **GNU/LINUX** con todas las herramientas y librerías indicadas en el apartado del entorno operacional.

En primer lugar se instalará la suite **ISE 14.3** mediante el instalador descargado directamente de la página **web de Xilinx [6]**. El producto que se ha instalado en nuestro caso es el sistema completo, aunque sería suficiente con instalar el producto *ISE Design Suite Embedded Edition* que incluye los programas que se van a utilizar. Posteriormente elegiremos las opciones de la instalación del producto.

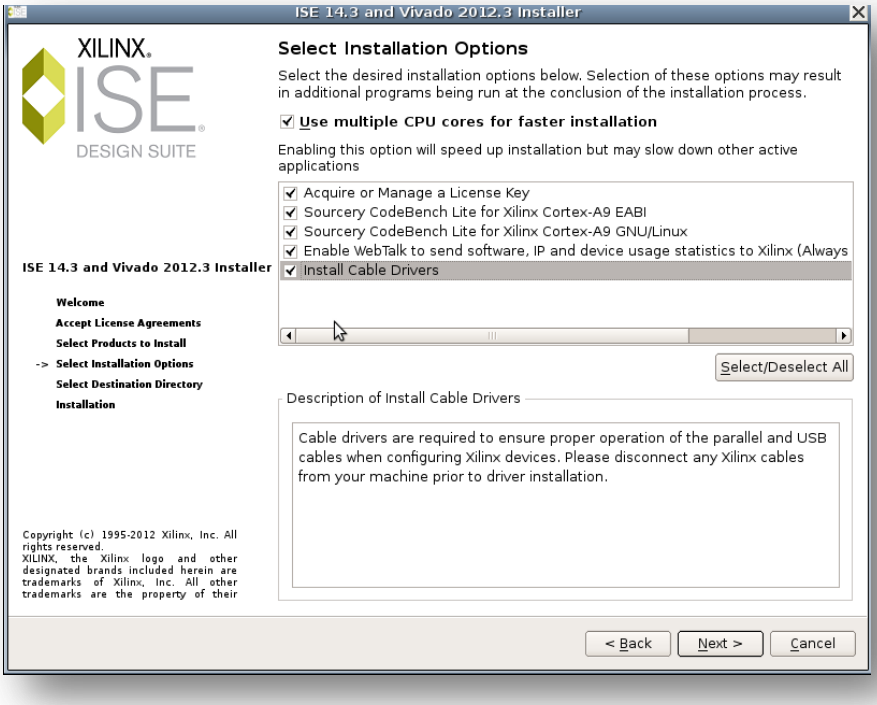


Ilustración 70: Instalación de ISE

Como muestra la ilustración, elegiremos que se quieren instalar **todas las herramientas** dentro del producto. En este paso es muy importante marcar la instalación de los "***cable drivers"***, ya que si no se instalan bien los drivers para conectar a la placa, usualmente mediante una conexión USB 2.0, los programas de la suite no serían capaces de poder conectarse con la Zedboard. También es útil en nuestro caso que se encargue de instalar las herramientas de **compilación de ARM**, las cuales van a ser necesarias para poder **compilar** código para el SoC.

Al terminar la instalación del ISE, si marcamos la casilla en las opciones, nos pedirá que **introduzcamos** la **licencia obtenida** para el uso de la suite. En el caso de que no hayamos marcado la casilla en la instalación, se pueden **gestionar las licencias** desde el menú de ayuda de los programas **XPS** o **ISE**.

La **licencia** se puede activar localizándola desde una **ruta local** como un archivo con extensión **".lic"** o en el caso de no tener licencia, obtener una licencia de evaluación o prueba. Para ello iremos al sitio Web donde podremos **generar** y **descargar** la **licencia**, siempre que la hayamos adquirido anteriormente.

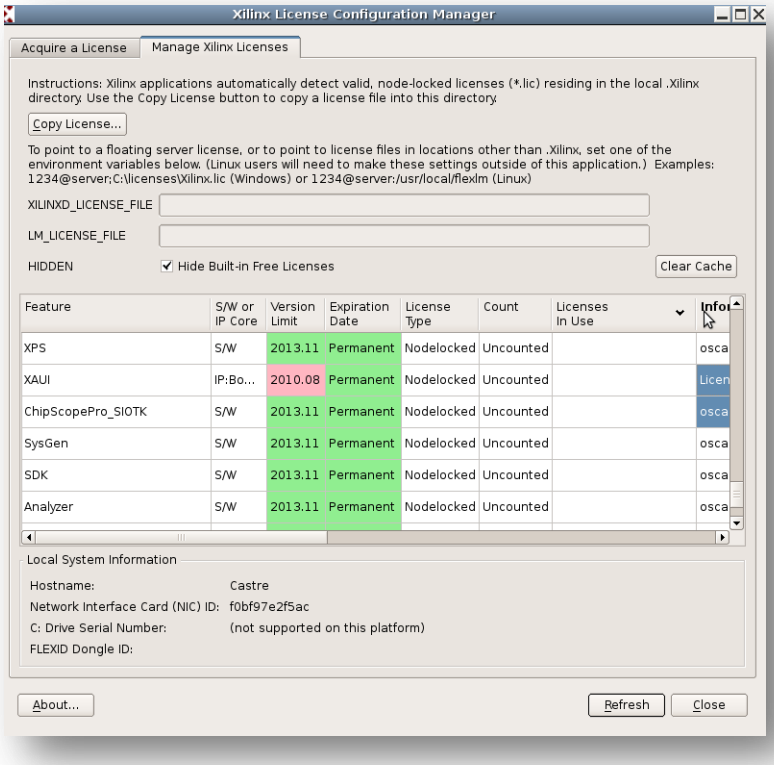


Ilustración 71: Gestión de Licencias ISE

Una vez **descargada** la **licencia** se podrá copiar al programa y **comprobar** las que tenemos **activas** en la pestaña “Manage Xilinx Licenses”.

Debido a que se va a utilizar la **última versión** de la suite de desarrollo **ISE 14.3**, la suite tiene algunos fallos todavía sin solucionar. Entre ellos hay que instalar un **parche** para las **librerías GCC** versión 6, publicado en los foros de Xilinx, el cual sobrescribe dos ficheros de la instalación **¡Error! No se encuentra el origen de la referencia.**.

Adicionalmente hay que lanzar el programa **cambiando** el **lenguaje** del entorno a **inglés**. Esto es debido a que si el lenguaje es el español, marca la delimitación de decimales como una coma y no como un punto. Para solucionarlo se puede cambiar el lenguaje del sistema operativo completo, o cambiar el lenguaje local del programa XPS.

Después de terminar de configurar toda la suite **se deberá comprobar** si están instalados correctamente los **drivers USB**. Para ello ser realizará una conexión a la placa mediante las interfaces **JTAG**, para programarla, y el **UART2USB** para comprobar la salida estándar en las pruebas.

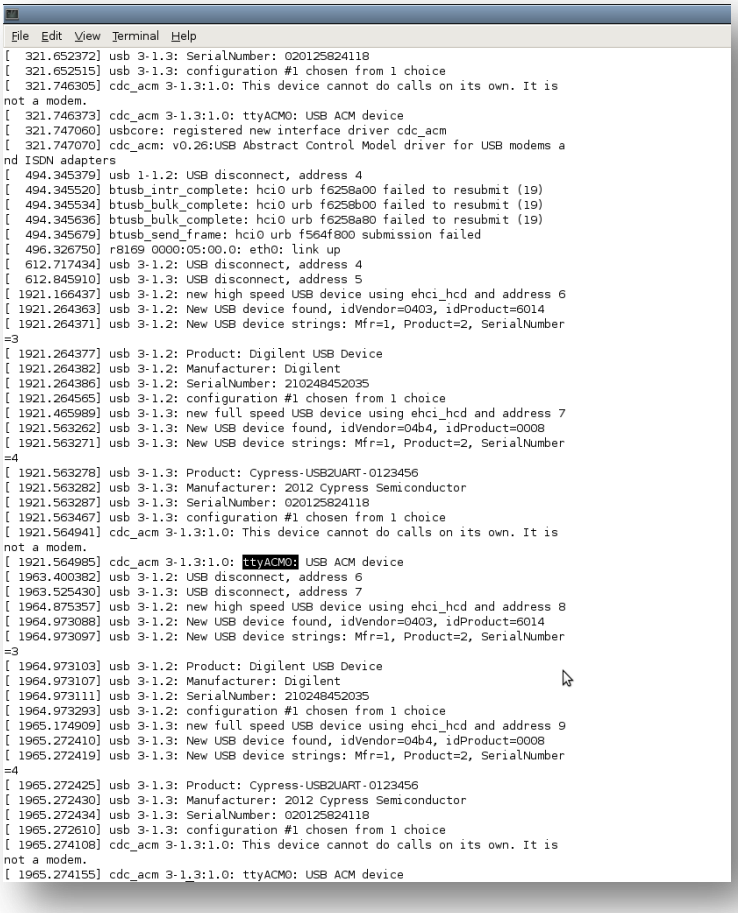


Ilustración 72: Salida comprobación periféricos dmesg

Para comprobarlo se utilizará el **comando** del SSOO ***dmesg***, que nos detallará el nombre de los **periféricos conectados**. Se encenderá la placa y **se buscará** las dos conexiones con las **interfaces USB**.

Finalmente, se introducirá el nombre de la **interfaz USB** donde vendrá la IO serial de la placa en un **terminal serial**. La SDK de Xilinx tiene una consola serial que puede ser configurada directamente, pero en nuestro caso se ha optado por un programa no perteneciente a la suite para ello, el **Gtkterm**.

11.2 Parámetros e interfaces axilite IPIF

En este anexo explicaremos el valor de los parámetros e interfaces que se encarga de gestionar el módulo axiliteIPIF

11.2.1 Parámetros

A continuación se enunciarán y explicarán las señales generales o parámetros que tiene el complemento para la facilidad de su uso.

- **C_S_AXI_DATA_WIDTH:** Esta constante define el tamaño de los datos que se pasarán por el bus. El valor es 32 debido a que el protocolo AXI4Lite no permite datos de mayor tamaño a 32 bits al ser una versión simplificada.
- **C_S_AXI_ADDR_WIDTH:** Esta constante define el tamaño de las direcciones de memoria que se utilizarán. Debido a que nuestro sistema tiene una arquitectura con direcciones de 32 bits, el valor será 32
- **C_S_AXI_MIN_SIZE:** Tamaño mínimo del rango de direcciones de memoria para el correcto funcionamiento del periférico.
- **C_FAMILY:** Parámetro que indica el modelo de FPGA al que pertenece el sistema donde se implementará el periférico.
- **C_USE_WSTRB:** Parámetro que indica si se quiere realizar la selección de bytes a escribir dentro de una palabra. El valor '1' indica que se quiere utilizar el selector, y el valor '0' que no se quiere utilizar.
- **C_DPHASE_TIMEOUT:** Número de ciclos máximos de espera al IP o lógica de usuario en un proceso de lectura/escritura.
- **C_ARD_ADDR_RANGE_ARRAY:** Rango de direcciones de memorias de la lógica de usuario, este dato se usa se quiere realizer un acceso a direcciones locales de memoria de la lógica de usuario.
- **C_ARD_NUM_CE_ARRAY:** Array que representa el número de chips/registros que se utilizarán en la señal de CE.

El periférico sumador sigue las directrices marcadas en el protocolo **AXI4LITE** de la **especificación AMBA** para comunicación mediante un bus del sistema **[14]**. Al ser el componente de más alto nivel del periférico, este componente es el encargado de tener las interfaces del bus, correspondientes a la especificación AXI4Lite, para realizar la conversión a otras señales más sencillas y realizar la gestión de memoria a la lógica del IP.

Como se ha comentado anteriormente, la funcionalidad del módulo es la de facilitar el trabajo a la hora de hacer un IP que trabaje sobre un bus y con direcciones de memoria. Para ello se facilitar la entrada/salida, codificando/decodificando las interfaces definidas por el protocolo del bus AXI4Lite, a otras interfaces de un uso más sencillo, y gestionando las direcciones de memoria. Por lo que además de las interfaces del bus, el componente también tiene interfaces simplificadas para el IP.

11.2.2 Interfaces bus AXI4Lite

Las siguientes interfaces son las **interfaces** de **entrada** y **salida** del bus especificadas por el protocolo:

- **Señales Globales:**
  - **ACLK**: Señal de **reloj** del bus.
  - **ARESETN**: Señal de **reset** global *active-Low*.
- **Señales de Escritura**:
  - **AWADDR**: Señal donde se indica la dirección de memoria de 32 bits donde se va a escribir.
  - **AWCACHE**: Señal que especifica el tipo de política de cache que se utiliza en la escritura.
  - **AWVALID:** Señal que indica si el dato de la señal AWADRR está disponible en el esclavo.
  - **AWREADY:** Señal que indica que el maestro está preparado para recibir la información de la señal AWADDR introducido.
  - **WDATA**: Señal de **32 bits** donde se indica el **dato** aescribir.
  - **WSTRB**: Señal que indica cuales de todos los **bytes** son **válidos**. Los dispositivos esclavos pueden decidir ignorar esta señal.
  - **WVALID**: Señal que indica si el dato WDATA está disponible.
  - **WREADY**: Señal que indica que el maestro está preparado para recibir la información de la señal WDATA introducido
  - **BRESP**: Esta señal indica el estado de la transacción de escritura. Las posibles respuestas son **OKAY**, **SLVERR** y **DECERR**.
  - **BVALID**: Señal que indica si el dato de la señal BRESP está disponible en el esclavo.
  - **BREADY**: Señal que indica que el maestro está preparado para recibir la información de la señal BREADY introducido.
- **Señales de Lectura**:
  - **ARADDR**: Señal donde se indica la dirección de memoria de 32 bits donde se va a leer.
  - **ARCACHE**: Señal que especifica el tipo de política de cache que se utiliza en la lectura.
  - **ARPROT**: Señal que especifica el tipo de política de protección de la lectura y si se trata de una lectura de datos o de instrucciones.
  - **ARVALID:** Señal que indica si el dato de la señal ARADRR está disponible en el esclavo.
  - **ARREADY:** Señal que indica que el maestro está preparado para recibir la información de la señal ARADDR introducido.
  - **RDATA**: Señal de **32 bits** donde se indica el **dato** que se desea leer.
  - **RSTRB**: Señal que indica cuales de todos los **bytes** son **válidos**. Los dispositivos esclavos pueden decidir ignorar esta señal.
  - **RVALID**: Señal que indica si el dato de la señal RDATA está disponible en el esclavo
  - **RREADY**: Señal que indica que el maestro está preparado para recibir la información de la señal RDATA introducido

11.2.3 Interfaces lógica de usuario / IP

A continuación se encuentran enumeradas y explicadas las interfaces que el módulo IPIF exporta para el desarrollo de un IP:

- **Bus2IP_Clk:** señal de reloj provista a la lógica de usuario. Esta señal es directamente la misma que la proveniente del bus.
- **Bus2IP_Resetn:** señal de salida de reset para la lógica de usuario *active-Low*.
- **IP2Bus_Data:** bus de entrada que indica el dato que se quiere leer en el IP, esto solo se hará si llega la señal de "IP2Bus_RdAck".
- **IP2Bus_WrAck:** señal de entrada active-High. Que indica si se va a realizar una escritura en el IP.
- **IP2Bus_RdAck:** señal de entrada active-High, que indica si se va a realizar una escritura en el IP.
- **IP2Bus_Error:** señal de entrada active-High, que indica si el IP ha tenido algún error en alguna de las operaciones solicitadas.
- **Bus2IP_Addr:** bus de salida que indica la dirección de memoria en la que se quiere hacer la lectura o la escritura.
- **Bus2IP_Data:** bus de salida que indica el dato que se quiere escribir en el IP, esto solo se hará si llega la señal de "IP2Bus_WrAck".
- **Bus2IP_RNW:** en esta señal de salida se indica el tipo de operación que se quiere realizar en el IP. El valor '1' corresponde al proceso de lectura y el valor '0' al proceso de escritura.
- **Bus2IP_BE:** esta señal de salida permite seleccionar a que bytes puede acceder el IP para una operación de lectura o escritura. El bit más significativo de la señal corresponde al byte más significativo del dato, y sucesivamente.
- **Bus2IP_CS:** bus de salida Active-High para la selección de chip del IP en el que se quiera hacer una operación. Cada bit del bus corresponde a un par de direcciones de memoria dentro del rango asignado. Es utilizado comúnmente para la implementación de memorias.
- **Bus2IP_RdCE:** bus de salida Active-High para la selección de chip del IP en el que se quiere hacer la lectura. Cada bit del bus corresponde a un chip de los chips definidos en el IP. Esta especialmente diseñado para realizar una lectura en un registro.
- **Bus2IP_WrCe:** bus de salida Active-High para la selección de chip del IP en el que se quiere hacer la escritura. Cada bit del bus corresponde a un chip de los chips definidos en el IP. Esta especialmente diseñado para realizar una escritura en un registro.

11.3 Código fuente de lógica de usuario

```
-- Autor -> Javier Castrejon Torrejon

------------------------------------------------------------------------------
--                  Logica de usuario del sumador
--
------------------------------------------------------------------------------

-- Importacion de librerias

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

------------------------------------------------------------------------------
-- Definicion de la Entidad
------------------------------------------------------------------------------
--   Constantes:
--   C_NUM_REG                -- Numero de registros disponibles
--   C_BWIDTH                         -- Ancho de banda del bus de datos
--
--   Interfaces recibidas de la librería axi_lite_ipif
--   Bus2IP_Clk              -- Señal de reloj del Bus al IP
--   Bus2IP_Reset            -- Señal de reset del Bus al IP
--   Bus2IP_Data              -- Puerto de datos del Bus al IP
--   Bus2IP_BE               -- Puerto del Bus al IP que permite la escritura de bytes
--   Bus2IP_RdCE              -- Chip enable de lectura del Bus al IP
--   Bus2IP_WrCE              -- Chip enable de escritura del Bus al IP
--   IP2Bus_Data             -- Bus de datos del IP al Bus
--   IP2Bus_RdAck             -- IP to Bus read transfer acknowledgement
--   IP2Bus_WrAck             -- IP to Bus write transfer acknowledgement
--   IP2Bus_Error            -- Señal de error del IP al Bus
------------------------------------------------------------------------------

entity user_logic is
  -- Definicion de constantes
  generic
          (
          -- Numero de registros
          C_NUM_REG                 : integer          := 3;
          -- Ancho de banda del bus
   C_BWIDTH                         : integer          := 32
          );

 -- Definicion de puertos
 port
 (
  -- Puertos definidos por la librería de logiCore axi-lite ipif
   Bus2IP_Clk               : in  std_logic;
   Bus2IP_Resetn              : in  std_logic;
   Bus2IP_Data              : in  std_logic_vector(C_BWIDTH-1 downto 0);
   Bus2IP_BE                : in  std_logic_vector(C_BWIDTH/8-1 downto 0);
   Bus2IP_RdCE              : in  std_logic_vector(C_NUM_REG-1 downto 0);
   Bus2IP_WrCE              : in  std_logic_vector(C_NUM_REG-1 downto 0);
   IP2Bus_Data              : out std_logic_vector(C_BWIDTH-1 downto 0);
   IP2Bus_RdAck             : out std_logic;
   IP2Bus_WrAck             : out std_logic;
   IP2Bus_Error             : out std_logic
 );

 attribute MAX_FANOUT : string;
 attribute SIGIS : string;

 attribute SIGIS of Bus2IP_Clk    : signal is "CLK";
 attribute SIGIS of Bus2IP_Resetn : signal is "RST";

end entity user_logic;

------------------------------------------------------------------------------
```

```
-- Sección de arquitectura
------------------------------------------------------------------------

architecture IMP of user_logic is

  ---------------------------------------
  -- Definicion de señales y registros internos
  ---------------------------------------
  signal slv_reg_in_0                   : std_logic_vector(C_BWIDTH-1 downto 0);
  signal slv_reg_in_1                   : std_logic_vector(C_BWIDTH-1 downto 0);
  signal slv_reg_out                            : std_logic_vector(C_BWIDTH-1 downto 0);
  signal slv_reg_write_sel                      : std_logic_vector(2 downto 0);
  signal slv_reg_read_sel                       : std_logic_vector(2 downto 0);
  signal slv_ip2bus_data                                                        :
std_logic_vector(C_BWIDTH-1 downto 0);
  signal slv_read_ack                           : std_logic;
  signal slv_write_ack                          : std_logic;


------------------------------------------------------------------------
-- Sección de Implementacion
------------------------------------------------------------------------


begin

  --Indicamos que el selector de registros se obtiene mediante el puerto de chip enable del bus
  slv_reg_write_sel <= Bus2IP_WrCE(2 downto 0);
  slv_reg_read_sel  <= Bus2IP_RdCE(2 downto 0);

  --Indicamos que se va a escribir o leer en los registros si esta seleccionado alguno
  slv_write_ack     <= Bus2IP_WrCE(0) or Bus2IP_WrCE(1) or Bus2IP_WrCE(2);
  slv_read_ack      <= Bus2IP_RdCE(0) or Bus2IP_RdCE(1) or Bus2IP_RdCE(2);

  -- Ejecuta un proceso que se encarga de escribir datos en los registros cuando cambia la señal de reloj y de realizar la
suma si no se va a escribir en ningun registro
  ESCRITURA : process( Bus2IP_Clk ) is
  begin

            -- Ejecuta el codigo si encuentra un flanco de subida en la señal de reloj del bus
              if Bus2IP_Clk'event and Bus2IP_Clk = '1' then

            -- Comprueba si esta activa la señal de reset, en ese caso resetea todos los registros a 0
                if Bus2IP_Resetn = '0' then
                                        slv_reg_in_0 <= (others => '0');
                                        slv_reg_in_1 <= (others => '0');
                                        slv_reg_out  <= (others => '0');
                                -- Si no intenta actualizar el valor en el registro selecionado recibida por el bus
             else
                                -- Combrueba el registro que quiere leer
       case slv_reg_write_sel is
                                --Quiere escribir en el segundo registro de entrada
        when "100" =>
                                                -- Intenta escribir en cada byte correspondiente del bus si esta habilitado
          for byte_index in 0 to (C_BWIDTH/8)-1 loop
                                                        if ( Bus2IP_BE(byte_index) = '1' ) then
                                                                -- Escribe en el registro el Byte leido del bus de datos
                                                                slv_reg_in_0(byte_index*8+7 downto byte_index*8) <=
Bus2IP_Data(byte_index*8+7 downto byte_index*8);
            end if;
          end loop;
                                --Quiere escribir en el primer registro de entrada
        when "010" =>
                                                -- Intenta escribir en cada byte correspondiente del bus si esta habilitado
                                                for byte_index in 0 to (C_BWIDTH/8)-1 loop
                                                        if ( Bus2IP_BE(byte_index) = '1' ) then
                                                                -- Escribe en el registro el Byte leido del bus de datos
                                                                slv_reg_in_1(byte_index*8+7 downto byte_index*8) <=
Bus2IP_Data(byte_index*8+7 downto byte_index*8);
            end if;
                                                --  slv_reg_out = slv_reg_in_0 + slv_reg_in_1;
          end loop;
                                -- No se ha seleccionado ningun registro por lo que pasara a realizar la suma
```

```
        when "000" =>
                                        -- Intenta escribir en cada byte correspondiente del bus si esta habilitado
                                        for byte_index in 0 to (C_BWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
                                                slv_reg_out(byte_index*8+7 downto byte_index*8) <=
slv_reg_in_0(byte_index*8+7 downto byte_index*8) +
                                                slv_reg_in_1(byte_index*8+7 downto byte_index*8);
                                                end if;el
                                        end loop;
                                    when others => null;
      end case;
    end if;
  end if;

 end process ESCRITURA;

            -- proceso que se encarga de escribir el dato correspondiente en el bus de datos
             LECTURA : process( slv_reg_read_sel, slv_reg_in_0, slv_reg_in_1, slv_reg_out) is
            begin
 case slv_reg_read_sel is
            when "100" => slv_ip2bus_data <= slv_reg_in_0;
            when "010" => slv_ip2bus_data <= slv_reg_in_1;
            when "001" => slv_ip2bus_data <= slv_reg_out;
            when others => null;
 end case;
 end process LECTURA;


 -- Redireccionamos los datos de lectura obtenidos de la interfaz si se va a realizar una lectura
 -- Leemos si slv_read_ack = 1

 IP2Bus_Data  <= slv_ip2bus_data when slv_read_ack = '1' else
              (others => '0');

 --Redirigimos las señales de ACK locales por los puertos

 IP2Bus_WrAck <= slv_write_ack;
 IP2Bus_RdAck <= slv_read_ack;
 IP2Bus_Error <= '0';

end IMP;
```